intel

Microprocessor and Peripheral Handbook

Volume II Peripheral



Order Number: 230843-005

LITERATURE

To order Intel literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Toll Free Number:
(800) 548-4725*

Use the order blank on the facing page or call our **Toll Free** Number listed above to order literature. Remember to add your local sales tax and a 10% postage charge for U.S. and Canada customers, 20% for customers outside the U.S. Prices are subject to change.

1988 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	**PRICE IN U.S. DOLLARS
COMPLETE SET OF 8 HANDBOOKS Save $50.00 off the retail price of $175.00	**231003**	**$125.00**
AUTOMOTIVE HANDBOOK (Not included in handbook Set)	231792	**$20.00**
COMPONENTS QUALITY/RELIABILITY HANDBOOK (Available in July)	210997	**$20.00**
EMBEDDED CONTROLLER HANDBOOK (2 Volume Set)	210918	**$23.00**
MEMORY COMPONENTS HANDBOOK	210830	**$18.00**
MICROCOMMUNICATIONS HANDBOOK	231658	**$22.00**
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	**$25.00**
MILITARY HANDBOOK (Not included in handbook Set)	210461	**$18.00**
OEM BOARDS AND SYSTEMS HANDBOOK	280407	**$18.00**
PROGRAMMABLE LOGIC HANDBOOK	296083	**$18.00**
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	**$20.00**
PRODUCT GUIDE Overview of Intel's complete product lines	210846	**N/C**
DEVELOPMENT TOOLS CATALOG	280199	**N/C**
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	**N/C**
LITERATURE PRICE LIST List of Intel Literature	210620	**N/C**

*Good in the U.S. and Canada

**These prices are for the U.S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.

About Our Cover:

The microprocessor has changed the way we work and live. Future generations will equate its impact on society with that of the invention of the wheel. Like the hub of a wheel, the processor stands at the center, but its potential is realized only when coupled with carefully matched peripheral devices. With this in mind, Intel provides complete "computing platforms" of processors and peripherals.

intel

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, and consulting services. For more information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It includes factory repair services and worldwide field service offices providing hardware repair services, software support services, customer training classes, and consulting services.

HARDWARE SUPPORT SERVICES

Intel is committed to providing an international service support package through a wide variety of service offerings available from Intel Hardware Support.

SOFTWARE SUPPORT SERVICES

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and COMMENTS Magazine). Basic support includes updates and the subscription service. Contracts are sold in environments which represent product groupings (i.e., iRMX environment).

CONSULTING SERVICES

Intel provides field systems engineering services for any phase of your development or support effort. You can use our systems engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training, and customizing or tailoring an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, bitbus and LAN applications.

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, and consulting services. For more information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international organization and a breadth of programs to meet a variety of customer needs. As your right expert, Intel's customer support is quite extensive. It includes factory repair services and worldwide field service offices providing hardware repair service, software support service, customer training classes, and consulting services.

HARDWARE SUPPORT SERVICES

Intel is committed to providing an international service support package through a wide variety of service offerings available from Intel Hardware Support.

SOFTWARE SUPPORT SERVICES

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and COMMENTS Magazine). Basic support includes updates and the subscription service. Contracts are sold in environments which represent product groupings (i.e., iRMX environment).

CONSULTING SERVICES

Intel provides field systems engineering services for any phase of your development or support effort. You can use our systems engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training, and customizing or tailoring an Intel product to providing technical and management consultation. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, buses and LAN applications.



Intel the Microcomputer Company:

When Intel invented the microprocessor in 1971, it created the era of microcomputers. Whether used as microcontrollers in automobiles or microwave ovens, or as personal computers or supercomputers, Intel's microcomputers have always offered leading-edge technology. In the second half of the 1980s, Intel architectures have held at least a 75% market share of microprocessors at 16 bits and above. Intel continues to strive for the highest standards in memory, microcomputer components, modules, and systems to give its customers the best possible competitive advantages.

MICROPROCESSOR AND PERIPHERAL HANDBOOK

VOLUME II PERIPHERAL

1988

intel

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, FASTPATH, GENIUS, i, î, ICE, iCEL, iCS, iDBP, iDIS, I²ICE, iLBX, i_m, iMDDX, iMMX, Inboard, Insite, Intel, int$_e$l, int$_e$lBOS, Intel Certified, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAP-NET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PC-BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, SupportNET, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix, 4-SITE.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

© INTEL CORPORATION 1987

Table of Contents

Alphanumeric Index ix

CHAPTER 1
Memory Controllers
DATA SHEETS
8203 64K Dynamic RAM Controller 1-1
8206 Error Detection and Correction Unit 1-17
8207 Dual-Port Dynamic RAM Controller 1-39
82C08 CHMOS Dynamic RAM Controller 1-86
APPLICATION NOTES
Interfacing the 8207 Dynamic RAM Controller to the 80186 AP-167 1-115
Interfacing the 8207 Advanced Dynamic RAM Controller to the 80286 AP-168 1-121

CHAPTER 2
Support Peripherals
DATA SHEETS
8231A Arithmetic Processing Unit 2-1
8253/8253-5 Programmable Interval Timer 2-14
8254 Programmable Interval Timer 2-25
82C54 CHMOS Programmable Interval Timer 2-46
8255A/8255A-5 Programmable Peripheral Interface 2-63
82C55A CHMOS Programmable Peripheral Interface 2-87
8256AH Multifunction Microprocessor Support Controller 2-110
8279/8279-5 Programmable Keyboard/Display Interface 2-134
82389 Message Passing Coprocessor, A MULTIBUS™ Bus Interface Controller 2-150
APPLICATION NOTES
Designing with the 8256 AP-153 2-171
8256AH Simplifies Computer I/O Design AP-183 2-242

CHAPTER 3
Floppy Disk Controllers
DATA SHEETS
8272A Single/Double Density Floppy Disk Controller 3-1
82072 CHMOS High Integration Floppy Disk Controller 3-32
APPLICATION NOTES
An Intelligent Data Base System Using the 8272 AP-116 3-77
Software Design and Implementation of Floppy Disk Systems AP-121 3-118
Designing with the 82072 CHMOS Single-Chip Floppy Disk Controller AP-289 3-166

CHAPTER 4
Hard Disk Controllers
DATA SHEETS
82064 Winchester Disk Controller with On-Chip Error Detection and Correction 4-1
APPLICATION NOTES
Multimode™ Winchester Controller Using the 82064 AP-402 4-33

CHAPTER 5
Universal Peripheral Interface Slave Microcontrollers
DATA SHEETS
UPI™-452 CHMOS Programmable I/O Processor (80/83/87452) 5-1
UPI™-41, 42: 8041AH/8042AH/8741AH/8742AH Universal Peripheral Interface 8-Bit Slave Microcontroller 5-55
8243 MCS-48 Input/Output Expander 5-74
APPLICATION NOTES
Applications Using the 8042 UPI™ Microcontroller 5-80
Complex Peripheral Control with the UPI™-42 AP-161 5-84

Table of Contents (Continued)

An 8741AH/8041A Digital Cassette Controller AP-90 5-139
UPI™-452 Accelerates iAPX 286 Bus Performance AP-281 5-146
SYSTEM SUPPORT
ICE™-42 8042 In-Circuit Emulator 5-166
iUP-200A/iUP-201A Universal PROM Programmers 5-174

CHAPTER 6
Alphanumeric Terminal Controllers
DATA SHEETS
8275H Programmable CRT Controller 6-1
8276H Small System CRT Controller 6-32
APPLICATION NOTES
A Low Cost CRT Terminal Using the 8275 AP-62 6-49

CHAPTER 7
Development Tools
SOFTWARE DEVELOPMENT TOOLS
8051 Software Development Packages 7-1
8096 Software Development Packages 7-9
8086/80186 Software Packages 7-19
286 Software Development Tools 7-40
Pascal 286 Software Package 7-45
PL/M Software Package 7-48
386 Software Tools 7-52
8087 Support Library 7-65
80287 Support Library 7-69
VAX/VMS Resident 8086/8088/80186 Software Development Package 7-73
VAX/VMS Resident Software Development Package for 80286 7-81
AEDIT Source Code and Text Editor for PC-DOS 7-87
PSCOPE High Level Program Debugger and for iRMX, XENIX, Series III and Series IV 7-90
PSCOPE 86 for DOS High-Level Application Program Debugger 7-98
PMON 386 PSCOPE Monitor 386 7-105
DMON 386 Debug Monitor 386 7-109
iPAT 86/88 Performance Analysis Tool 7-112
iPAT 286 Performance Analysis Tool 7-123
IN-CIRCUIT EMULATORS
I²ICE Integrated In-Circuit Emulation System 7-134
ICE-386 In-Circuit Emulator for the 8386 7-146
iSBE-96 Development Kit Single Board Emulator and Assembler 7-157
VLSiCE-96 In-Circuit Emulator for the 8X9X Family 7-165
ICE-5100/452 In-Circuit Emulator for the UPI-452 Family 7-175
ICE-5100/044 In-Circuit Emulator for the RUPI-44 Family 7-183
ICE-5100/252 In-Circuit Emulator for the MCS-51 Family 7-191
FACT SHEETS
ICE-186 In-Circuit Emulator 7-201
ICE-286 In-Circuit Emulator 7-204
Real-Time Transparent 80C196 In-Circuit Emulator 7-207

Alphanumeric Index

286 Software Development Tools 7-40
386 Software Tools 7-52
80287 Support Library 7-69
8051 Software Development Packages 7-1
8086/80186 Software Packages 7-19
8087 Support Library 7-65
8096 Software Development Packages 7-9
82C08 CHMOS Dynamic RAM Controller 1-86
82C54 CHMOS Programmable Interval Timer 2-46
82C55A CHMOS Programmable Peripheral Interface 2-87
8203 64K Dynamic RAM Controller 1-1
8206 Error Detection and Correction Unit 1-17
82064 Winchester Disk Controller with On-Chip Error Detection and Correction 4-1
8207 Dual-Port Dynamic RAM Controller 1-39
82072 CHMOS High Integration Floppy Disk Controller 3-32
8231A Arithmetic Processing Unit 2-1
82389 Message Passing Coprocessor, A MULTIBUS™ Bus Interface Controller 2-150
8243 MCS-48 Input/Output Expander 5-74
8253/8253-5 Programmable Interval Timer 2-14
8254 Programmable Interval Timer 2-25
8255A/8255A-5 Programmable Peripheral Interface 2-63
8256AH Multifunction Microprocessor Support Controller 2-110
8256AH Simplifies Computer I/O Design AP-183 2-242
8272A Single/Double Density Floppy Disk Controller 3-1
8275H Programmable CRT Controller 6-1
8276H Small System CRT Controller 6-32
8279/8279-5 Programmable Keyboard/Display Interface 2-134
A Low Cost CRT Terminal Using the 8275 AP-62 6-49
An Intelligent Data Base System Using the 8272 AP-116 3-77
An 8741AH/8041A Digital Cassette Controller AP-90 5-139
Applications Using the 8042 UPI™ Microcontroller 5-80
AEDIT Source Code and Text Editor for PC-DOS 7-87
Complex Peripheral Control with the UPI™-42 AP-161 5-84
Designing with the 82072 CHMOS Single-Chip Floppy Disk Controller AP-289 3-166
Designing with the 8256 AP-153 2-171
DMON 386 Debug Monitor 386 7-109
I^2ICE Integrated In-Circuit Emulation System 7-134
iPAT 286 Performance Analysis Tool 7-123
iPAT 86/88 Performance Analysis Tool 7-112
iSBE-96 Development Kit Single Board Emulator and Assembler 7-157
iUP-200A/iUP-201A Universal PROM Programmers 5-174
Interfacing the 8207 Advanced Dynamic RAM Controller to the 80286 AP-168 1-121
Interfacing the 8207 Dynamic RAM Controller to the 80186 AP-167 1-115
ICE™-42 8042 In-Circuit Emulator 5-166
ICE-186 In-Circuit Emulator 7-201
ICE-286 In-Circuit Emulator 7-204
ICE-386 In-Circuit Emulator for the 8386 7-146
ICE-5100/044 In-Circuit Emulator for the RUPI-44 Family 7-183
ICE-5100/252 In-Circuit Emulator for the MCS-51 Family 7-191
ICE-5100/452 In-Circuit Emulator for the UPI-452 Family 7-175
Multimode™ Winchester Controller Using the 82064 AP-402 4-33
Pascal 286 Software Package 7-45
PL/M Software Package 7-48
PMON 386 PSCOPE Monitor 386 7-105

Alphanumeric Index (Continued)

PSCOPE High Level Program Debugger and for iRMX, XENIX,—Series III and Series IV . . . 7-90
PSCOPE 86 for DOS High-Level Application Program Debugger 7-98
Real-Time Transparent 80C196 In-Circuit Emulator 7-207
Software Design and Implementation of Floppy Disk Systems AP-121 3-118
UPI™-41, 42: 8041AH/8042AH/8741AH/8742AH Universal Peripheral Interface 8-Bit Slave Microcontroller 5-55
UPI™-452 Accelerates iAPX 286 Bus Performance AP-281 5-146
UPI™-452 CHMOS Programmable I/O Processor (80/83/87452) 5-1
VAX/VMS Resident Software Development Package for 80286 7-81
VAX/VMS Resident 8086/8088/80186 Software Development Package 7-73
VLSiCE-96 In-Circuit Emulator for the 8X9X 7-165

INTRODUCTION

Intel microprocessors and peripherals provide a complete solution in increasingly complex application environments. Quite often, a single peripheral device will replace anywhere from 20 to 100 TTL devices (and the associated design time that goes with them).

Built-in functions and standard Intel microprocessor/peripheral interface deliver very real *time* and *performance* advantages to the designer of microprocessor-based systems.

REDUCED TIME TO MARKET

When you can purchase an off-the-shelf solution that replaces a number of discrete devices, you're also replacing all the design, testing, and debug *time* that goes with them.

INCREASED RELIABILITY

At Intel, the rate of failure for devices is carefully tracked. Highest reliability is a tangible goal that translates to higher reliability for your product, reduced downtime, and reduced repair costs. And as more and more functions are intergrated on a single VLSI device, the resulting system requires less power, produces less heat, and requires fewer mechanical connections—again resulting in greater system reliability.

LOWER PRODUCTION COST

By minimizing design time, increasing reliability, and replacing numerous parts, microprocessor and peripheral solutions can contribute dramatically to lower product costs.

HIGHER SYSTEM PERFORMANCE

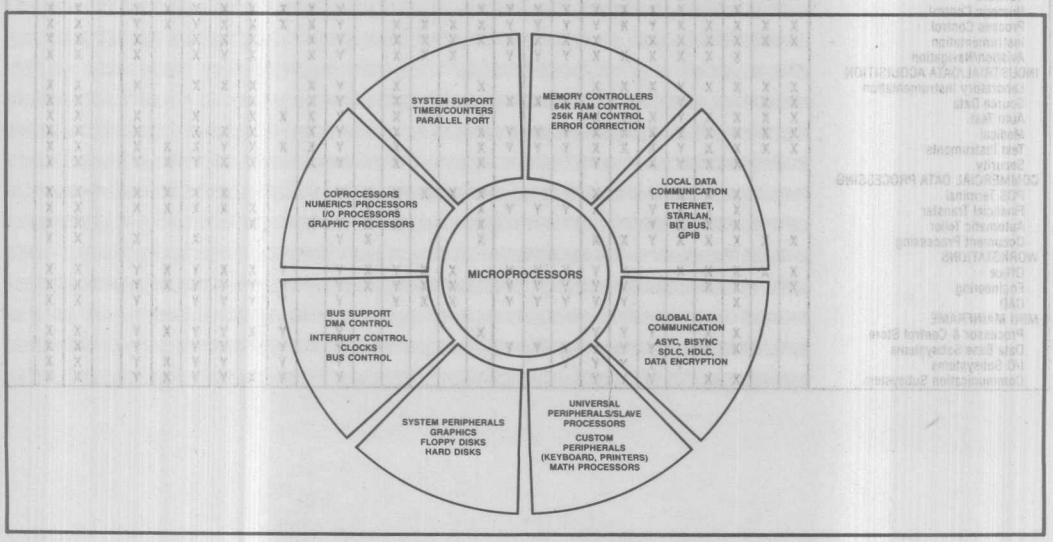
Intel microprocessors and peripherals provide the highest system performance for the demands of today's (and tomorrow's) microprocessor-based applications. For example, the 80386 32 bit offers the highest performance for multitasking, multiuser systems. Intel's peripheral products have been designed with the future in mind. They support all of Intel's 8, 16 and 32 bit processors.

HOW TO USE THE GUIDE

The following application guide illustrates the range of microprocessors and peripherals that can be used for the applictions in the vertical column of the left. The peripherals are grouped by the I/O function they control. CRT datacommunication, universal (user programmable), mass storage dynamic RAM controllers, and CPU/bus support.

An "X" in a horizontal application row indicates a potential peripheral or CPU, depending upon the features desired. For example, a conversational terminal could use either of the three display controllers, depending upon features like the number of characters per row or font capability. A "Y" indicates a likely candidate, for example, the 8272A Floppy Disk Controller in a small business computer.

The Intel microprocessor and peripherals family provides a broad range of time-saving, high performance solutions.



APPLICATION CHART

POTENTIAL APPLICATION X — TYPICAL APPLICATION Y

APPLICATION	MICROPROCESSORS								NUMERIC PROCESSORS			SUPPORT								DMA				CACHE	DRAM			
	8088	80C88	8086	80C86	80188	80186	80286	80386	8087	80287	80387	8231	8253	8254/C54	82555A/C55A	8256AH	8259A	8279	82389	8237	82258	82380	82560	82385	8203	8206	8207	82C08
PERIPHERALS																												
Printers	X	X			X	Y		X					X	X	X	X	Y			Y	X	X		X	X	X	X	X
Plotters	X	X	X	X	X	Y		X							X		Y			X	X	X		X	X	X	X	X
Keyboards					X	Y									X			X										
MASS STORAGE																												
Hard Disk	X	X	X	X	Y	Y									X				X	Y	Y							
Mini Winchester	X	X			Y	Y														Y	Y							
Tape					X	Y									X					Y	Y							
Cassette					Y	Y									X					Y								
Floppy/Mini					Y	Y														Y								
COMMUNICATIONS																												
Digital Telephone																												
ISDN					Y																							
PBX					X	X	Y	Y					X	X	X	X	Y			X	Y	Y		Y	X	X	X	X
LANs	X	X	X	X	X	Y	Y						X	X			Y			X	Y	X	X					
Modems																		X							X			X
Bisync													X	X														
SDLC/HDLC													X	X														
Serial Backplane																	Y			X		X	X					
Central Office			X	X		X	Y	Y					X	X		X	Y		X	X	Y	Y		Y	X	X	X	X
Network Control			Y			X	Y	Y					X	X			Y		X		Y	Y	X	Y	X	X	X	X
OFFICE/BUS																												
Copier/FAX	X	X			X	Y										X	Y	X		X	X		X		X	X	X	X
Wordprocessor	X	X	X	X	X	X	Y	Y					Y	Y	Y	X	Y			Y		Y		Y	X	X	X	X
Typewriter					X										Y		Y			Y			X		X	X	X	X
Electronic Mail			X		X	X		X									Y			Y	X	X		X	X	X	X	X
Transaction System			Y		X	X	X	X		X	X		Y	Y	X	X	Y					X		X	X	X	X	X
Data Entry	X	X	X	X	X	X							Y	Y		X	Y	X		X					X	X	X	X
COMPUTERS																												
Small Business Computer			X	X	Y	X	X	Y	Y	Y	Y		Y	Y	Y	X	Y			X	X	Y	X	Y	X	X	X	X
PC	Y	Y	Y	Y		X	X	Y	Y	X	X		Y	Y	Y	X	Y			Y		Y	X	Y	X	X	X	X
Portable PC		Y		Y		X		X	X	X	X		Y	Y	Y	X	Y			Y		X	X	X	X			X
Home Computer	X	X	X	X	Y	X	X	X					Y	Y	Y	X	Y			Y		X		X	X		X	X
TERMINALS																												
Conversational					Y								Y	Y		X				X						X	X	
Graphics CRT			Y	Y		Y	Y	X	Y	Y	Y		Y	Y	Y	X	Y				X	X		X		X	X	
Editing	X	X	X	X	X	X	X	X					Y	Y	Y	Y	Y			X		X		X		X	X	
Intelligent	X	X	X	X	Y	Y		X					Y	Y	Y	X	Y			Y		X	X	X		X	X	
Integrated Voice/Data	X	X						X														X		X				
Videotex	X	X	X	X	X	X		X					Y	Y	Y		Y			Y	X	X		X		X	X	
Printing: Laser, Impact	X	X	X	X	X	X		X		X	X		Y	Y	Y		Y			Y	X	X	X	X		X	X	
Portable	X	Y	X	Y	Y			X					Y	Y	Y		Y			X		X		X				
INDUSTRIAL/AUTO																												
Robotics					Y	Y	X	Y	Y	Y	Y	X			Y		X	X	X	X	X	Y	X	Y		X	X	
Network	X	X	X	X	X	X	X	Y									Y		X	X	Y	Y	X	Y		X	X	
Numeric Control			X		X	X	X	Y	Y	Y	Y	X					Y	X	X	X	X	Y	X	Y		X	X	
Process Control	X	X	X	X	X	Y	X	Y	Y	Y	Y	X	X	X	X		Y	X	X	X	X	Y	X	Y		X	X	
Instrumentation	X	X	X	X	X	X		X	X	X	X	X	X	X	X		Y	X		X	X	X		X		X	X	
Aviation/Navigation			X	X	X	X	X	X	Y	Y	Y		X	X	X		Y	X		X	Y	X		X		X	X	
INDUSTRIAL/DATA ACQUISITION																												
Laboratory Instrumentation	X	X	X	X	Y	X	X	X				X			X		Y	X		X	X	X		X		X	X	
Source Data	X	X			Y				X	X	X				X		Y	X		X						X	X	
Auto Test	X	X	X	X	Y	X	Y	X				X			X		Y	X	X	X		X		X		X	X	
Medical	X	X	X	X	Y	X	X	X	Y	Y	Y	X			X		Y	X		X	X	X		X		X	X	
Test Instruments	X	X	X	X	Y	X	X	X	Y	Y	Y	X			X		Y	X	X	X	Y	X	X	X		X	X	
Security			X	X	Y	X		Y				X			X		Y	X		X	X	Y	X	Y		X	X	
COMMERCIAL DATA PROCESSING																												
POS Terminal			X	X	X	Y		X				X	X	X	X	X	Y				X	X	X	X		X	X	
Financial Transfer			X	X	X	Y		X	Y	Y	Y	X			X	X	Y			X	X	X	X	X		X	X	
Automatic Teller			X	X	X	Y						X			X	X	Y	X		X						X	X	
Document Processing	X	X	X	X	X	Y	X	X				X				X	Y					X		X		X	X	
WORKSTATIONS																												
Office	X	X	X	X	X		Y	Y	Y	Y	Y		X	X	Y	X	Y		Y	X	X	Y	X	Y		X	X	
Engineering	X		X	X			Y	Y	Y	Y	Y		X	X	Y	X	Y		Y	Y	Y	Y	X	Y		X	X	
CAD			X				Y	Y	Y	Y	Y		X	X	Y		Y		Y	Y	Y	Y		Y		X	X	
MINI MAINFRAME																												
Processor & Control Store			X	X		Y	Y	Y				X					Y		Y	X	Y	Y	X	Y		X	X	
Data Base Subsystems			X	X		Y	Y	Y	Y	Y	Y						Y		Y		Y	Y		Y		X	X	
I/O Subsystems					Y	Y	Y	Y	Y								Y		Y		Y	Y	X	Y		X	X	
Communication Subsystem			X	X		Y	Y	Y									Y		Y	X	Y	Y	X	Y		X	X	

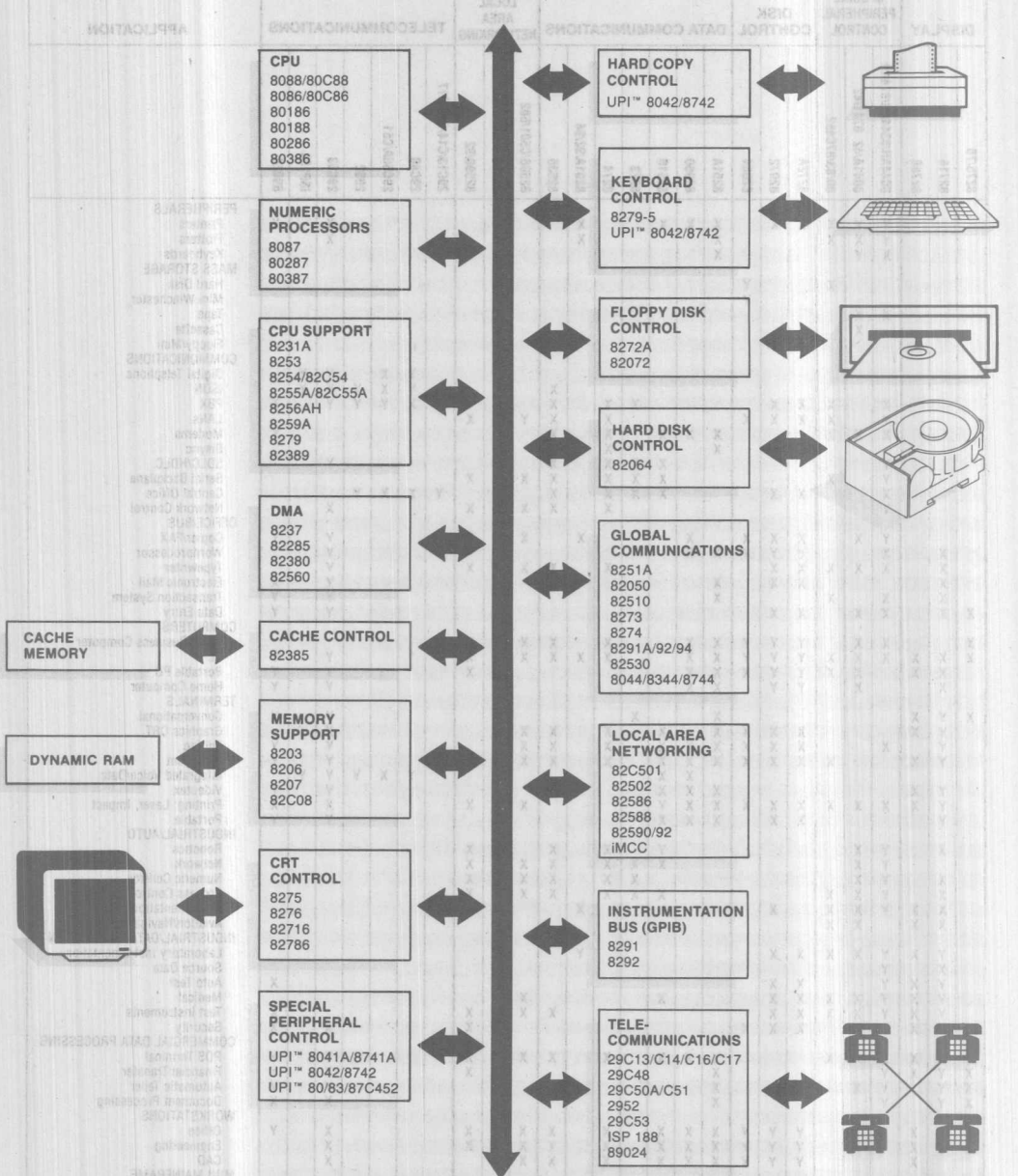
APPLICATION CHART

POTENTIAL APPLICATION X — TYPICAL APPLICATION Y

DISPLAY			SPECIAL PERIPHERAL CONTROL			DISK CONTROL			DATA COMMUNICATIONS						LOCAL AREA NETWORKING			TELECOMMUNICATIONS							APPLICATION
8275/76	82716	82786	8044AH/8344AH/8744H	8041A/42 8741/42	80/83/87C452	8272A	82072	82064	8251A	82050	82510	8273	8274	8291A/92/94	82530	82586/C501/502	82588	82590/92	29C13/C14/C16/C17	29C48	29C50A/C51	2952	29C53	ISP188	89024	
			Y	Y	X	X	X		X	X	X			X									Y		X	PERIPHERALS
			Y	X	X				X					X									X			Printers
			X	Y					X																	Plotters
																										Keyboards
					X			Y																		MASS STORAGE
								Y																		Hard Disk
			X	X																						Mini Winchester
				X																						Tape
						Y	Y																			Cassette
																										Floppy/Mini
																				Y	X		Y	Y		COMMUNICATIONS
															X					Y	X	Y	Y	Y		Digital Telephone
			X		X	X	X					X	X		X				Y	X	Y	Y	Y		X	ISDN
			Y		X	X	X	X					X		X	Y	X	X								PBX
			X	X	X				X	X	X		X		X										Y	LANs
									X				X		X											Modems
			Y									X	X		X								Y			Bisync
			Y		X							X	X		X	X	X	X								SDLC/HDLC
			X			X	X					X	X		X				Y	X	X	Y				Serial Backplane
			Y										X		X	X	X	X					X			Central Office
																										Network Control
			Y	X		X	X	X						X		X	X	X					Y		Y	OFFICE/BUS
	X		X			Y	Y	Y	X	X													Y			Copier/FAX
	X		X	X	X	X	X						X		X	X	X	X					Y			Wordprocessor
	X					X	X		X														Y		X	Typewriter
	X		X		X																		Y		Y	Electronic Mail
	X		X	X		X	X		X														Y		Y	Transaction System
X	X		X	X		X	X																Y		Y	Data Entry
X			X	X		Y	Y	Y	X	X			X		X	X	X	X								COMPUTERS
X	X	X	X	X	X	Y	Y	Y	X	X			X	X	X	X	X	X					Y		Y	Small Business Computer
	X	X		X	X	Y	Y	X	X	X							X	X					Y		Y	PC
	X			X		Y	Y		X	X													Y		Y	Portable PC
X	Y	X							X			X											Y		Y	Home Computer
	Y	X				X	X	X	X		X		X		X	X							X			TERMINALS
	Y	X				X	X	X	X	X	X		X		X	X							X		Y	Conversational
	Y		X			X	X	X	X	X	X		X		X	X							Y		Y	Graphics CRT
	Y	X			X	X	X	X	X	X	X		X		X	X	X	X		Y	X	Y	Y	Y	Y	Editing
	Y	X																								Intelligent
										X	X															Integrated Voice/Data
	Y	X							X	X	X															Videotex
	Y	X	X	X	X	X	X	X	X	X	Y					X	X	X					X		X	Printing: Laser, Impact
	Y					X	X		X	X	X												Y		Y	Portable
	X		Y	X							Y		X		X											INDUSTRIAL/AUTO
			Y	X							X	X	X		X	X										Robotics
	X		Y	X								X	X		X	X										Network
	X		Y	X	X							X	X		X	X										Numeric Control
	X	X	Y	X	X		X				X	X	X	X	X											Process Control
	X	X		X	X																					Instrumentation
	Y	X	Y	X	X									Y												Aviation/Navigation
	Y	X	Y	X	X	X	X																			INDUSTRIAL/DATA ACQUISITION
	X		Y																							Laboratory Instrumentation
	Y	X	Y			X	X									X									X	Source Data
	Y	X	Y	X	X	X	X				X					X									X	Auto Test
	Y	X	Y	X	X	X	X								X	X									X	Medical
	Y	X	Y	X																						Test Instruments
																										Security
	X	X	X	X	X				X	X	X		X		X	X	X	X					Y		Y	COMMERCIAL DATA PROCESSING
X	Y	X	Y						X								X	X					X		Y	POS Terminal
X	Y		Y	X					X														X		Y	Financial Transfer
X	Y		Y						X														Y		Y	Automatic Teller
	Y	X	X	X		Y	Y	Y	X	X	X		X		X	X	X	X					X		Y	Document Processing
	X	X		X		Y	Y	Y	X	X	X		X		X	X	X	X					X			WORKSTATIONS
		X				Y	Y	Y	X	X	X		X		X	X							X			Office
																										Engineering
Y						X	X						X		X		X	X								CAD
X								X															X			MINI MAINFRAME
			X		X					X	X		X		X		X	X					X			Processor & Control Store
			X		X				X	X	X		X		X		X	X					Y		X	Data Base Subsystems
																										I/O Subsystems
																										Communication Subsystem

Get Your Kit Together!

Intel's Microsystem Components Kit Solution



© Intel 1987

October 1987
Order Number: 230664-006

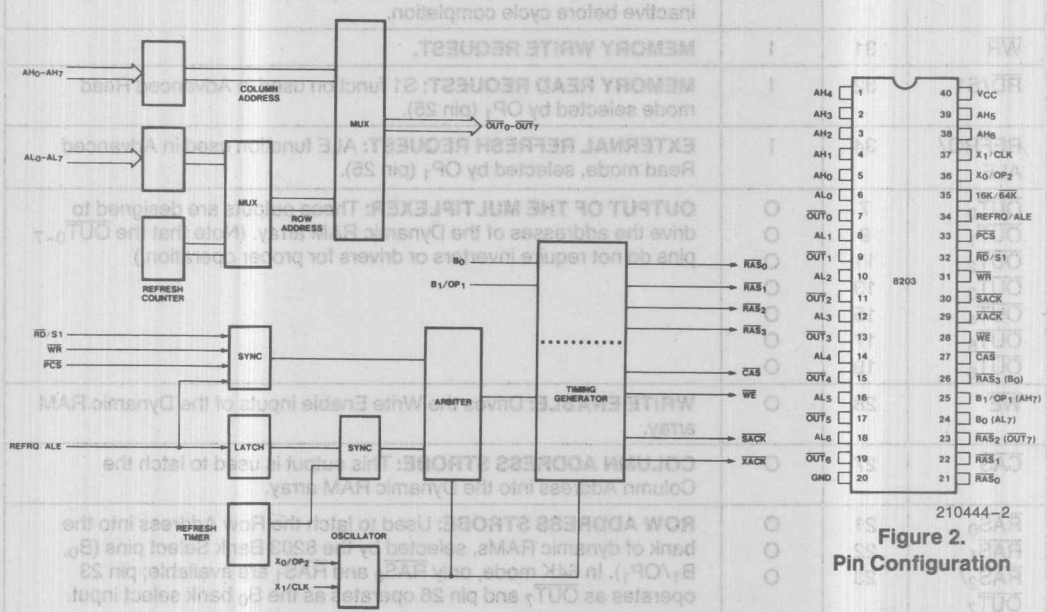
Memory Controllers 1

1 Memory Controllers

8203
64K DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to Control 64K and 16K Dynamic Memories**
- **Directly Addresses and Drives Up to 64 Devices Without External Drivers**
- **Provides Address Multiplexing and Strobes**
- **Provides a Refresh Timer and a Refresh Counter**
- **Provides Refresh/Access Arbitration**
- **Internal Clock Capability with the 8203-1 and the 8203-3**
- **Fully Compatible with Intel® 8080A, 8085A, iAPX88, and iAPX 86 Family Microprocessors**
- **Decodes CPU Status for Advanced Read Capability in 16K Mode with the 8203-1 and the 8203-3.**
- **Provides System Acknowledge and Transfer Acknowledge Signals**
- **Refresh Cycles May be Internally or Externally Requested (For Transparent Refresh)**
- **Internal Series Damping Resistors on All RAM Outputs**

The Intel® 8203 is a Dynamic RAM System Controller designed to provide all signals necessary to use 64K or 16K Dynamic RAMs in microcomputer systems. The 8203 provides multiplexed addresses and address strobes, refresh logic, refresh/access arbitration. Refresh cycles can be started internally or externally. The 8203-1 and the 8203-3 support an internal crystal oscillator and Advanced Read Capability. The 8203-3 is a ±5% V_{CC} part.



210444–1

Figure 1. 8203 Block Diagram

210444–2

Figure 2. Pin Configuration

November 1987
Order Number: 210444-006

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
AL_0 AL_1 AL_2 AL_3 AL_4 AL_5 AL_6	6 8 10 12 14 16 18	I I I I I I I	**ADDRESS LOW:** CPU address inputs used to generate memory row address.
AH_0 AH_1 AH_2 AH_3 AH_4 AH_5 AH_6	5 4 3 2 1 39 38	I I I I I I I	**ADDRESS HIGH:** CPU address inputs used to generate memory column address.
B_0/AL_7 $B_1/OP_1/$ AH_7	24 25	I I	**BANK SELECT INPUTS:** Used to gate the appropriate $\overline{RAS}$ output for a memory cycle. B_1/OP_1 option used to select the Advanced Read Mode. (Not available in 64K mode.) See Figure 5. When in 64K RAM Mode, pins 24 and 25 operate as the AL_7 and AH_7 address inputs.
$\overline{PCS}$	33	I	**PROTECTED CHIP SELECT:** Used to enable the memory read and write inputs. Once a cycle is started, it will not abort even if $\overline{PCS}$ goes inactive before cycle completion.
$\overline{WR}$	31	I	**MEMORY WRITE REQUEST.**
$\overline{RD}/S1$	32	I	**MEMORY READ REQUEST:** S1 function used in Advanced Read mode selected by OP_1 (pin 25).
REFRQ/ ALE	34	I	**EXTERNAL REFRESH REQUEST:** ALE function used in Advanced Read mode, selected by OP_1 (pin 25).
$\overline{OUT}_0$ $\overline{OUT}_1$ $\overline{OUT}_2$ $\overline{OUT}_3$ $\overline{OUT}_4$ $\overline{OUT}_5$ $\overline{OUT}_6$	7 9 11 13 15 17 19	O O O O O O O	**OUTPUT OF THE MULTIPLEXER:** These outputs are designed to drive the addresses of the Dynamic RAM array. (Note that the $\overline{OUT}_{0-7}$ pins do not require inverters or drivers for proper operation.)
$\overline{WE}$	28	O	**WRITE ENABLE:** Drives the Write Enable inputs of the Dynamic RAM array.
$\overline{CAS}$	27	O	**COLUMN ADDRESS STROBE:** This output is used to latch the Column Address into the Dynamic RAM array.
$\overline{RAS}_0$ $\overline{RAS}_1$ $\overline{RAS}_2/$ $\overline{OUT}_7$ $\overline{RAS}_3/B_0$	21 22 23 26	O O O I/O	**ROW ADDRESS STROBE:** Used to latch the Row Address into the bank of dynamic RAMs, selected by the 8203 Bank Select pins (B_0, B_1/OP_1). In 64K mode, only $\overline{RAS}_0$ and $\overline{RAS}_1$ are available; pin 23 operates as $\overline{OUT}_7$ and pin 26 operates as the B_0 bank select input.
$\overline{XACK}$	29	O	**TRANSFER ACKNOWLEDGE:** This output is a strobe indicating valid data during a read cycle or data written during a write cycle. $\overline{XACK}$ can be used to latch valid data from the RAM array.

Table 1. Pin Descriptions (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{SACK}$	30	O	**SYSTEM ACKNOWLEDGE:** This output indicates the beginning of a memory access cycle. It can be used as an advanced transfer acknowledge to eliminate wait states. (Note: If a memory access request is made during a refresh cycle, $\overline{SACK}$ is delayed until $\overline{XACK}$ in the memory access cycle).
X_0/OP_2 X_1/CLK	36 37	I/O I/O	**OSCILLATOR INPUTS:** These inputs are designed for a quartz crystal to control the frequency of the oscillator. If X_0/OP_2 is shorted to pin 40 (V_{CC}) or if X_0/OP_2 is connected to +12V through a 1 KΩ resistor then X_1/CLK becomes a TTL input for an external clock. (Note: Crystal mode for the 8203-1 and the 8203-3 only).
16K/$\overline{64K}$	35	I	**MODE SELECT:** This input selects 16K mode or 64K mode. Pins 23–26 change function based on the mode of operation.
V_{CC}	40		**POWER SUPPLY:** +5V.
GND	20		**GROUND.**

FUNCTIONAL DESCRIPTION

The 8203 provides a complete dynamic RAM controller for microprocessor systems as well as expansion memory boards.

The 8203 has two modes, one for 16K dynamic RAMs and one for 64Ks, controlled by pin 35.

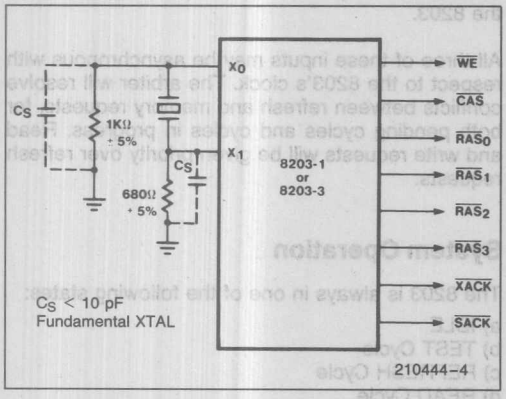


Figure 3. Crystal Operation for the 8203-1 and 8203-3

All 8203 timing is generated from a single reference clock. This clock is provided via an external oscillator or an on-chip crystal oscillator. All output signal transitions are synchronous with respect to this clock reference, except for the trailing edges of the CPU handshake signals $\overline{SACK}$ and $\overline{XACK}$.

CPU memory requests normally use the $\overline{RD}$ and $\overline{WR}$ inputs. The Advanced-Read mode allows ALE and S1 to be used in place of the $\overline{RD}$ input.

Failsafe refresh is provided via an internal timer which generates refresh requests. Refresh requests can also be generated via the REFRQ input.

An on-chip synchronizer/arbiter prevents memory and refresh requests from affecting a cycle in progress. The READ, WRITE, and external REFRESH requests may be asynchronous to the 8203 clock; on-chip logic will synchronize the requests, and the arbiter will decide if the requests should be delayed, pending completion of a cycle in progress.

16K/64 Option Selection

Pin 35 is a strap input that controls the two 8203 modes. Figure 4 shows the four pins that are multiplexed. In 16K mode (pin 35 tied to V_{CC} or left open), the 8203 has two Bank Select inputs to select one of four $\overline{RAS}$ outputs. In this mode, the 8203 is exactly compatible with the Intel 8202A Dynamic RAM Controller. In 64K mode (pin 35 tied to GND), there is only one Bank Select input (pin 26) to select the two $\overline{RAS}$ outputs. More than two banks of 64K dynamic RAMs can be used with external logic.

Other Option Selections

The 8203 has two strapping options. When OP_1 is selected (16K mode only), pin 32 changes from a $\overline{RD}$ input to an S1 input, and pin 34 changes from a REFRQ input to an ALE input. See "Refresh Cycles" and "Read Cycles" for more detail. OP_1 is selected by tying pin 25 to +12V through a 5.1 KΩ resistor on the 8203-1 or 8203-3 only.

When OP_2 is selected, the internal oscillator is disabled and pin 37 changes from a crystal input (X_1) to a CLK input for an external TTL clock. OP_2 is selected by shorting pin 36 (X_0/OP_2) directly to pin 40 (V_{CC}). No current limiting resistor should be used. OP_2 may also be selected by tying pin 36 to +12V through a 1 KΩ resistor.

Refresh Timer

The refresh timer is used to monitor the time since the last refresh cycle occurred. When the appropriate amount of time has elapsed, the refresh timer will request a refresh cycle. External refresh requests will reset the refresh timer.

Refresh Counter

The refresh counter is used to sequentially refresh all of the memory's rows. The 8-bit counter is incremented after every refresh cycle.

Pin #	16K Function	64K Function
23	$\overline{RAS}_2$	Address Output ($\overline{OUT}_7$)
24	Bank Select (B_0)	Address Input (AL_7)
25	Bank Select (B_1)	Address Input (AH_7)
26	$\overline{RAS}_3$	Bank Select (B_0)

Figure 4. 16K/64K Mode Selection

	Inputs		Outputs			
	B_1	B_0	$\overline{RAS}_0$	$\overline{RAS}_1$	$\overline{RAS}_2$	$\overline{RAS}_3$
16K Mode	0	0	0	1	1	1
	0	1	1	0	1	1
	1	0	1	1	0	1
	1	1	1	1	1	0
64K Mode	—	0	0	1	—	—
	—	1	1	0	—	—

Figure 5. Bank Selection

Address Multiplexer

The address multiplexer takes the address inputs and the refresh counter outputs, and gates them onto the address outputs at the appropriate time. The address outputs, in conjunction with the $\overline{RAS}$ and $\overline{CAS}$ outputs, determine the address used by the dynamic RAMs for read, write, and refresh cycles. During the first part of a read or write cycle, AL_0–AL_7 are gated to $\overline{OUT}_0$–$\overline{OUT}_7$, then AH_0–AH_7 are gated to the address outputs.

During a refresh cycle, the refresh counter is gated onto the address outputs. All refresh cycles are $\overline{RAS}$-only refresh ($\overline{CAS}$ inactive, $\overline{RAS}$ active).

To minimize buffer delay, the information on the address outputs is inverted from that on the address inputs.

$\overline{OUT}_0$–$\overline{OUT}_7$ do not need inverters or buffers unless additional drive is required.

Synchronizer/Arbiter

The 8203 has three inputs, REFRQ/ALE (pin 34), $\overline{RD}$ (pin 32) and $\overline{WR}$ (pin 31). The $\overline{RD}$ and $\overline{WR}$ inputs allow an external CPU to request a memory read or write cycle, respectively. The REFRQ/ALE input allows refresh requests to be requested external to the 8203.

All three of these inputs may be asynchronous with respect to the 8203's clock. The arbiter will resolve conflicts between refresh and memory requests, for both pending cycles and cycles in progress. Read and write requests will be given priority over refresh requests.

System Operation

The 8203 is always in one of the following states:

a) IDLE
b) TEST Cycle
c) REFRESH Cycle
d) READ Cycle
e) WRITE Cycle

The 8203 is normally in the IDLE state. Whenever one of the other cycles is requested, the 8203 will

Description	Pin #	Normal Function	Option Function
B_1/OP_1 (16K only)/AH_7	25	Bank ($\overline{RAS}$) Select	Advanced-Read Mode (8203-1, -3)
X_0/OP_2	36	Crystal Oscillator (8203-1 and 8203-3)	External Oscillator

Figure 6. 8203 Option Selection

leave the IDLE state to perform the desired cycle. If no other cycles are pending, the 8203 will return to the IDLE state.

Test Cycle

The TEST Cycle is used to check operation of several 8203 internal functions. TEST cycles are requested by activating the $\overline{PCS}$, $\overline{RD}$ and $\overline{WR}$ inputs. The TEST Cycle will reset the refresh address counter and perform a WRITE Cycle. The TEST Cycle should not be used in normal system operation, since it would affect the dynamic RAM refresh.

Refresh Cycles

The 8203 has two ways of providing dynamic RAM refresh:

1) Internal (failsafe) refresh
2) External (hidden) refresh

Both types of 8203 refresh cycles activate all of the $\overline{RAS}$ outputs, while $\overline{CAS}$, $\overline{WE}$, $\overline{SACK}$, and $\overline{XACK}$ remain inactive.

Internal refresh is generated by the on-chip refresh timer. The timer uses the 8203 clock to ensure that refresh of all rows of the dynamic RAM occurs every 2 milliseconds (128 cycles) or every 4 milliseconds (256 cycles). If $\overline{REFRQ}$ is inactive, the refresh timer will request a refresh cycle every 10–16 microseconds.

External refresh is requested via the $\overline{REFRQ}$ input (pin 34). External refresh control is not available when the Advanced-Read mode is selected. External refresh requests are latched, then synchronized to the 8203 clock.

The arbiter will allow the refresh request to start a refresh cycle only if the 8203 is not in the middle of a cycle.

When the 8203 is in the idle state a simultaneous memory request and external refresh request will result in the memory request being honored first. This 8203 characteristic can be used to "hide" refresh cycles during system operation. A circuit similar to Figure 7 can be used to decode the CPU's instruction fetch status to generate an external refresh request. The refresh request is latched while the 8203 performs the instruction fetch; the refresh cycle will start immediately after the memory cycle is completed, even if the $\overline{RD}$ input has not gone inactive. If the CPU's instruction decode time is long enough, the 8203 can complete the refresh cycle before the next memory request is generated.

If the 8203 is not in the idle state then a simultaneous memory request and an external refresh request may result in the refresh request being honored first.

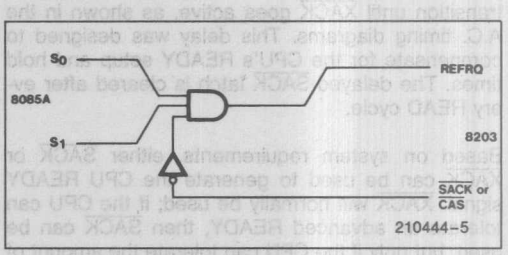


Figure 7. Hidden Refresh

Certain system configurations require complete external refresh requests. If external refresh is requested faster than the minimum internal refresh timer (t_{REF}), then, in effect, all refresh cycles will be caused by the external refresh request, and the internal refresh timer will never generate a refresh request.

Read Cycles

The 8203 can accept two different types of memory Read requests:

1) Normal Read, via the $\overline{RD}$ input
2) Advanced Read, using the S1 and ALE inputs (16K mode only)

The user can select the desired Read request configuration via the B_1/OP_1 hardware strapping option on pin 25.

	Normal Read	Advanced Read
Pin 25	B_1 Input	OP_1 (+12V)
Pin 32	RD Input	S1 Input
Pin 34	REFRQ Input	ALE Input
# RAM Banks	4 ($\overline{RAS}_{0-3}$)	2 ($\overline{RAS}_{2-3}$)
Ext. Refresh	Yes	No

Figure 8. 8203 Read Options

Normal Reads are requested by activating the $\overline{RD}$ input, and keeping it active until the 8203 responds with an $\overline{XACK}$ pulse. The $\overline{RD}$ input can go inactive as soon as the command hold time (t_{CHS}) is met.

Advanced Read cycles are requested by pulsing ALE while S1 is active; if S1 is inactive (low) ALE is ignored. Advanced Read timing is similar to Normal Read timing, except the falling edge of ALE is used as the cycle start reference.

If a Read cycle is requested while a refresh cycle is in progress, then the 8203 will set the internal delayed-$\overline{SACK}$ latch. When the Read cycle is eventually started, the 8203 will delay the active $\overline{SACK}$ transition until $\overline{XACK}$ goes active, as shown in the A.C. timing diagrams. This delay was designed to compensate for the CPU's READY setup and hold times. The delayed-$\overline{SACK}$ latch is cleared after every READ cycle.

Based on system requirements, either $\overline{SACK}$ or $\overline{XACK}$ can be used to generate the CPU READY signal. $\overline{XACK}$ will normally be used; if the CPU can tolerate an advanced READY, then $\overline{SACK}$ can be used, but only if the CPU can tolerate the amount of advance provided by $\overline{SACK}$. If $\overline{SACK}$ arrives too early to provide the appropriate number of WAIT states, then either $\overline{XACK}$ or a delayed form of $\overline{SACK}$ should be used.

Write Cycles

Write cycles are similar to Normal Read cycles, except for the $\overline{WE}$ output. $\overline{WE}$ is held inactive for Read cycles, but goes active for Write cycles. All 8203 Write cycles are "early-write" cycles; $\overline{WE}$ goes active before $\overline{CAS}$ goes active by an amount of time sufficient to keep the dynamic RAM output buffers turned off.

General System Considerations

All memory requests (Normal Reads, Advanced Reads, Writes) are qualified by the $\overline{PCS}$ input. $\overline{PCS}$ should be stable, either active or inactive, prior to the leading edge of $\overline{RD}$, $\overline{WR}$, or ALE. Systems which use battery backup should pullup $\overline{PCS}$ to prevent erroneous memory requests.

In order to minimize propagation delay, the 8203 uses an inverting address multiplexer without latches. The system must provide adequate address setup and hold times to guarantee $\overline{RAS}$ and $\overline{CAS}$ setup and hold times for the RAM. The t_{AD} A.C. parameter should be used for this system calculation.

The B_0–B_1 inputs are similar to the address inputs in that they are not latched. B_0 and B_1 should not be changed during a memory cycle, since they directly control which $\overline{RAS}$ output is activated.

The 8203 uses a two-stage synchronizer for the memory request inputs ($\overline{RD}$, $\overline{WR}$, ALE), and a separate two stage synchronizer for the external refresh input (REFRQ). As with any synchronizer, there is always a finite probability of metastable states inducing system errors. The 8203 synchronizer was designed to have a system error rate less than 1 memory cycle every three years based on the full operating range of the 8203.

A microprocessor system is concerned when the data is valid after $\overline{RD}$ goes low. See Figure 9. In order to calculate memory read access times, the dynamic RAM's A.C. specifications must be examined, especially the $\overline{RAS}$-access time (t_{RAC}) and the $\overline{CAS}$-access time (t_{CAC}). Most configurations will be $\overline{CAS}$-access limited; i.e., the data from the RAM will be stable $t_{cc,max}$ (8203) + t_{CAC} (RAM) after a memory read cycle is started. Be sure to add any delays (due to buffers, data latches, etc.) to calculate the overall read access time.

Since the 8203 normally performs "early-write" cycles, the data must be stable at the RAM data inputs by the time $\overline{CAS}$ goes active, including the RAM's data setup time. If the system does not normally guarantee sufficient write data setup, you must either delay the $\overline{WR}$ input signal or delay the 8203 $\overline{WE}$ output.

Delaying the $\overline{WR}$ input will delay all 8203 timing, including the READY handshake signals, $\overline{SACK}$ and $\overline{XACK}$, which may increase the number of WAIT states generated by the CPU.

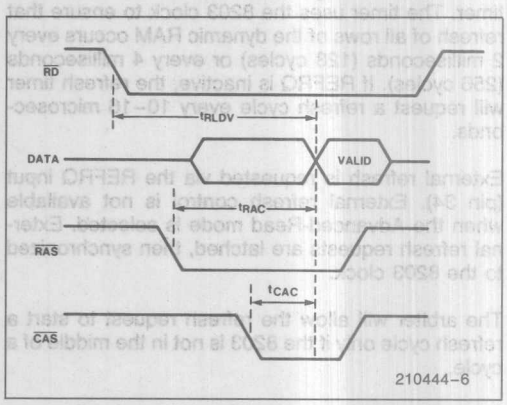


Figure 9. Read Access Time

If the $\overline{WE}$ output is externally delayed beyond the $\overline{CAS}$ active transition, then the RAM will use the falling edge of $\overline{WE}$ to strobe the write data into the RAM. This $\overline{WE}$ transition should not occur too late during the $\overline{CAS}$ active transition, or else the $\overline{WE}$ to $\overline{CAS}$ requirements of the RAM will not be met.

The $\overline{RAS}_{0-3}$, $\overline{CAS}$, $\overline{OUT}_{0-7}$, and $\overline{WE}$ outputs contain on-chip series damping resistors (typically 20Ω) to minimize overshoot.

Some dynamic RAMs require more than 2.4V V_{IH}. Noise immunity may be improved for these RAMs by adding pull-up resistors to the 8203's outputs. Intel RAMs do not require pull-up resistors.

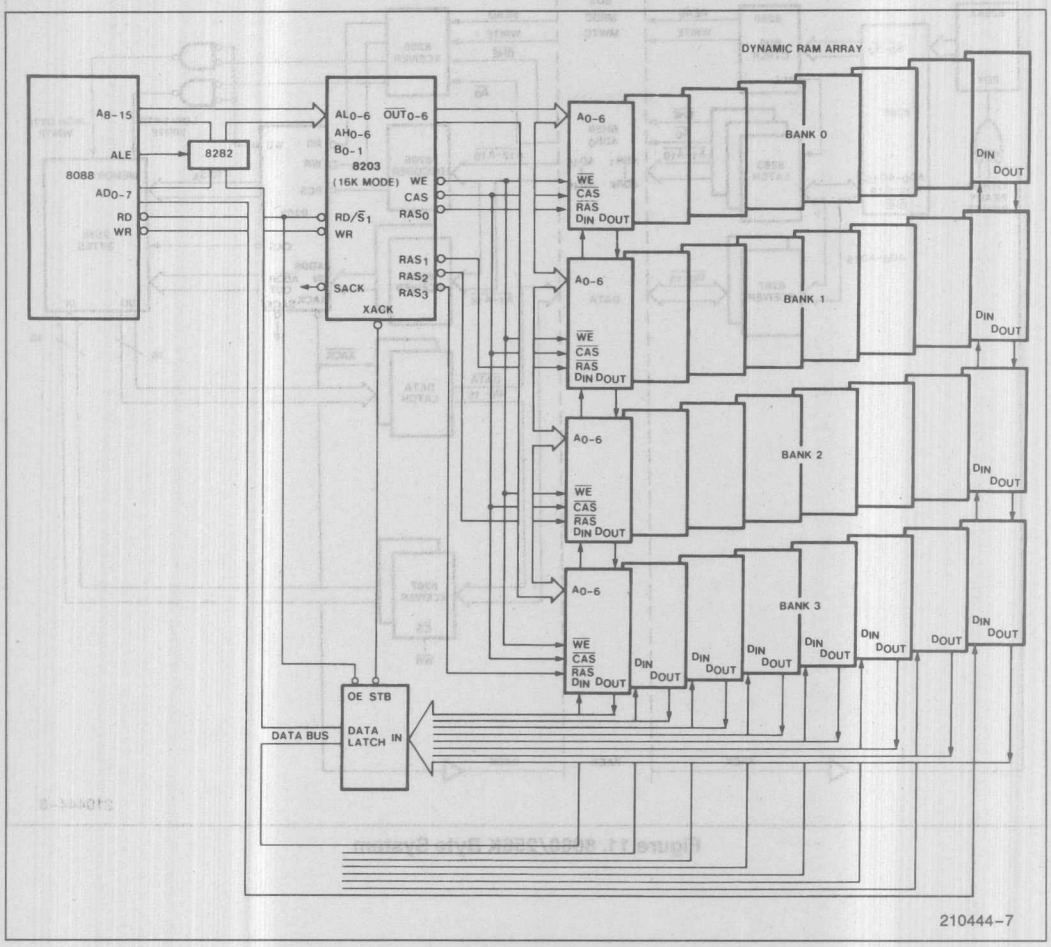


Figure 10. Typical 8088 System

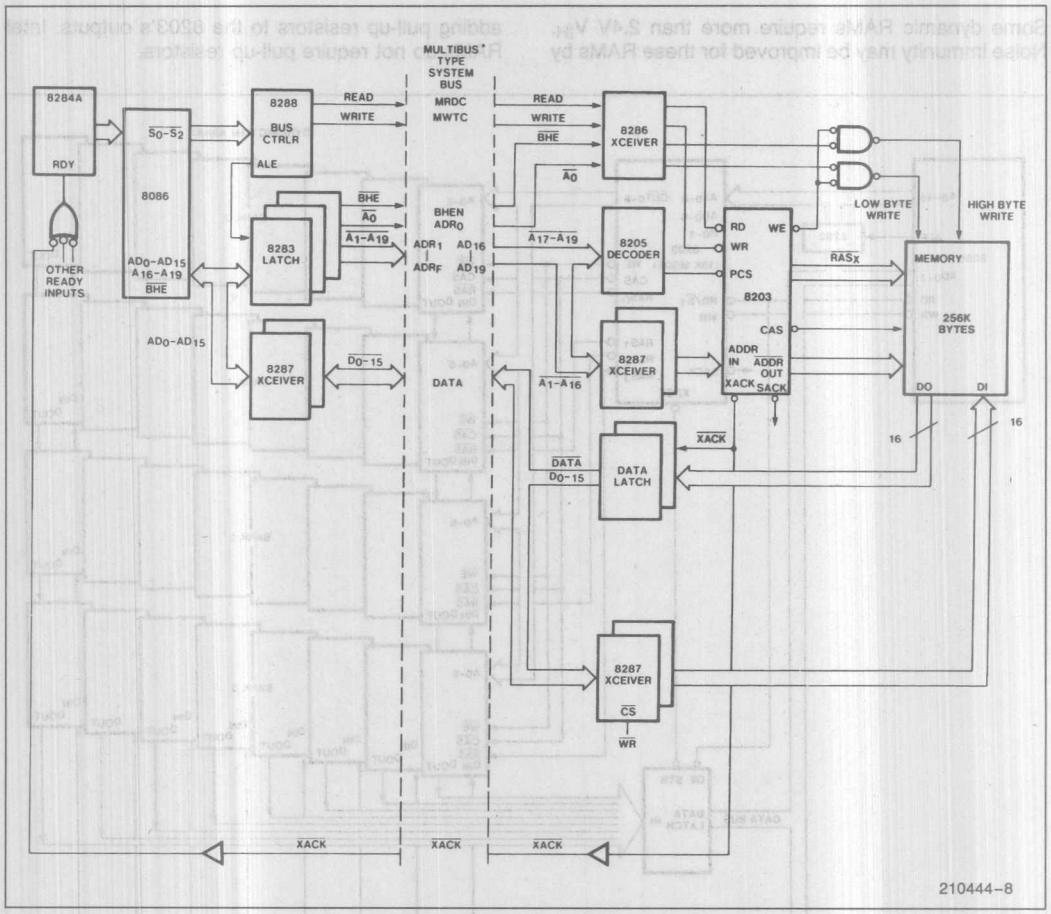


Figure 11. 8086/256K Byte System

ABSOLUTE MAXIMUM RATINGS*

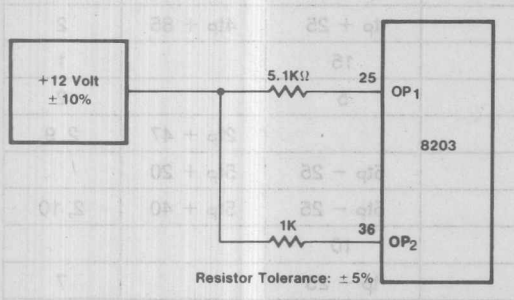
Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On Any Pin
With Respect to Ground .....−0.5V to +7V
Power Dissipation .....1.6 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5.0V ±10% (5.0V ±5% for 8203-3); GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{CC}	Input Clamp Voltage		−1.0	V	I_C = −5 mA
I_{CC}	Power Supply Current		290	mA	
I_F	Forward Input Current				
	CLK, 64K/16K Mode Select		−2.0	mA	V_F = 0.45V
	All Other Inputs(3)		−320	μA	V_F = 0.45V
I_R	Reverse Input Current(3)		40	μA	$V_R = V_{CC}$(1, 5)
V_{OL}	Output Low Voltage				
	$\overline{SACK}$, $\overline{XACK}$		0.45	V	I_{OL} = 5 mA
	All Other Outputs		0.45	V	I_{OL} = 3 mA
V_{OH}	Output High Voltage				V_{IL} = 0.65V
	$\overline{SACK}$, $\overline{XACK}$	2.4		V	I_{OH} = −1 mA
	All Other Outputs	2.6		V	I_{OH} = −1 mA
V_{IL}	Input Low Voltage		0.8	V	V_{CC} = 5.0V(2)
V_{IH1}	Input High Voltage	2.0	V_{CC}	V	V_{CC} = 5.0V
V_{IH2}	Option Voltage		V_{CC}	V	(4)
C_{IN}	Input Capacitance		30	pF	F = 1 MHz(6)
					V_{BIAS} = 2.5V, V_{CC} = 5V

NOTES:
1. I_R = 200 μA for pin 37 (CLK).
2. For test mode $\overline{RD}$ & $\overline{WR}$ must be held at GND.
3. Except for pin 36 in XTAL mode.
4. 8203-1 and 8203-3 support both OP_1 and OP_2, 8203 only supports OP_2.



210444-3

5. I_R = 150 μA for pin 35 (Mode Select 16K/64K).
6. Sampled not 100% tested, T_A = 25°C.

A.C. CHARACTERISTICS

T_J = 0°C to 70°C; V_{CC} = 5V ±10% (5.0V ±5% for 8203-3); GND = 0V
Measurements made with respect to $\overline{RAS}_0$–$\overline{RAS}_3$, $\overline{CAS}$, $\overline{WE}$, $\overline{OUT}_0$–$\overline{OUT}_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in ns.

Symbol	Parameter	Min	Max	Notes
t_P	Clock Period	40	54	
t_{PH}	External Clock High Time	20		
t_{PL}	External Clock Low Time—Above (>) 20 MHz	17		
t_{PL}	External Clock Low Time—Below (≤) 20 MHz	20		
t_{RC}	Memory Cycle Time	$10t_P - 30$	$12t_P$	4, 5
t_{REF}	Refresh Time (128 cycles)	$264t_P$	$288t_P$	
t_{RP}	$\overline{RAS}$ Precharge Time	$4t_P - 30$		
t_{RSH}	$\overline{RAS}$ Hold After $\overline{CAS}$	$5t_P - 30$		3
t_{ASR}	Address Setup to $\overline{RAS}$	$t_P - 30$		3
t_{RAH}	Address Hold From $\overline{RAS}$	$t_P - 10$		3
t_{ASC}	Address Setup to $\overline{CAS}$	$t_P - 30$		3
t_{CAH}	Address Hold from $\overline{CAS}$	$5t_P - 20$		3
t_{CAS}	$\overline{CAS}$ Pulse Width	$5t_P - 10$		
t_{WCS}	$\overline{WE}$ Setup to $\overline{CAS}$	$t_P - 40$		
t_{WCH}	$\overline{WE}$ Hold After $\overline{CAS}$	$5t_P - 35$		8
t_{RS}	$\overline{RD}$, $\overline{WR}$, ALE, REFRQ Delay From $\overline{RAS}$	$5t_P$		2, 6
t_{MRP}	$\overline{RD}$, $\overline{WR}$ Setup to $\overline{RAS}$	0		5
t_{RMS}	REFRQ Setup to $\overline{RD}$, $\overline{WR}$	$2t_P$		6
t_{RMP}	REFRQ Setup to $\overline{RAS}$	$2t_P$		5
t_{PCS}	$\overline{PCS}$ Setup to $\overline{RD}$, $\overline{WR}$, ALE	20		
t_{AL}	S1 Setup to ALE	15		
t_{LA}	S1 Hold From ALE	30		
t_{CR}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{RAS}$ Delay	$t_P + 30$	$2t_P + 70$	2
t_{CC}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{CAS}$ Delay	$t_P + 25$	$4t_P + 85$	2
t_{SC}	CMD Setup to Clock	15		1
t_{MRS}	$\overline{RD}$, $\overline{WR}$ Setup to REFRQ	5		2
t_{CA}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{SACK}$ Delay		$2t_P + 47$	2, 9
t_{CX}	$\overline{CAS}$ to $\overline{XACK}$ Delay	$5t_P - 25$	$5t_P + 20$	
t_{CS}	$\overline{CAS}$ to $\overline{SACK}$ Delay	$5t_P - 25$	$5t_P + 40$	2, 10
t_{ACK}	$\overline{XACK}$ to $\overline{CAS}$ Setup	10		
t_{XW}	$\overline{XACK}$ Pulse Width	$t_P - 25$		7
t_{CK}	$\overline{SACK}$, $\overline{XACK}$ Turn-Off Delay		35	
t_{KCH}	CMD Inactive Hold After $\overline{SACK}$, $\overline{XACK}$	10		

A.C. CHARACTERISTICS (Continued)

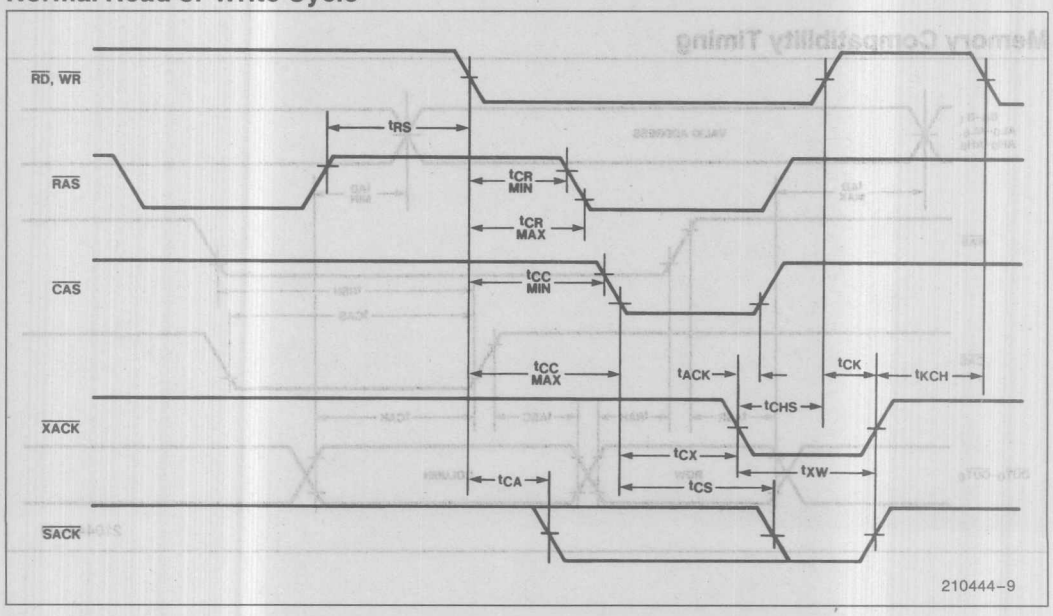
T_J = 0°C to 70°C; V_{CC} = 5V ±10% (5.0V ±5% for 8203-3); GND = 0V
Measurements made with respect to $\overline{RAS}_0$–$\overline{RAS}_3$, $\overline{CAS}$, $\overline{WE}$, $\overline{OUT}_0$–$\overline{OUT}_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in ns.

Symbol	Parameter	Min	Max	Notes
t_{LL}	REFRQ Pulse Width	20		
t_{CHS}	CMD Hold Time	30		11
t_{RFR}	REFRQ to $\overline{RAS}$ Delay		$4t_P + 100$	6
t_{WW}	$\overline{WR}$ to $\overline{WE}$ Delay	0	50	8
t_{AD}	CPU Address Delay	0	40	3

NOTES:
1. t_{SC} is a reference point only. ALE, $\overline{RD}$, $\overline{WR}$, and REFRQ inputs do not have to be externally synchronized to 8203 clock.
2. If t_{RS} min and t_{MRS} min are met then t_{CA}, t_{CR}, and t_{CC} are valid, otherwise t_{CS} is valid.
3. t_{ASR}, t_{RAH}, t_{ASC}, t_{CAH}, and t_{RSH} depend upon B_0–B_1 and CPU address remaining stable throughout the memory cycle. The address inputs are not latched by the 8203.
4. For back-to-back refresh cycles, t_{RC} max = 13 t_P.
5. t_{RC} max is valid only if t_{RMP} min is met (READ, WRITE followed by REFRESH) or t_{MRP} min is met (REFRESH followed by READ, WRITE).
6. t_{RFR} is valid only if t_{RS} min and t_{RMS} min are met.
7. t_{XW} min applies when $\overline{RD}$, $\overline{WR}$ has already gone high. Otherwise $\overline{XACK}$ follows $\overline{RD}$, $\overline{WR}$.
8. $\overline{WE}$ goes high according to t_{WCH} or t_{WW}, whichever occurs first.
9. t_{CA} applies only when in normal $\overline{SACK}$ mode.
10. t_{CS} applies only when in delayed $\overline{SACK}$ mode.
11. t_{CHS} must be met only to ensure a $\overline{SACK}$ active pulse when in delayed $\overline{SACK}$ mode. $\overline{XACK}$ will always be activated for at least t_{XW} (t_P − 25 ns). Violating t_{CHS} min does not otherwise affect device operation.

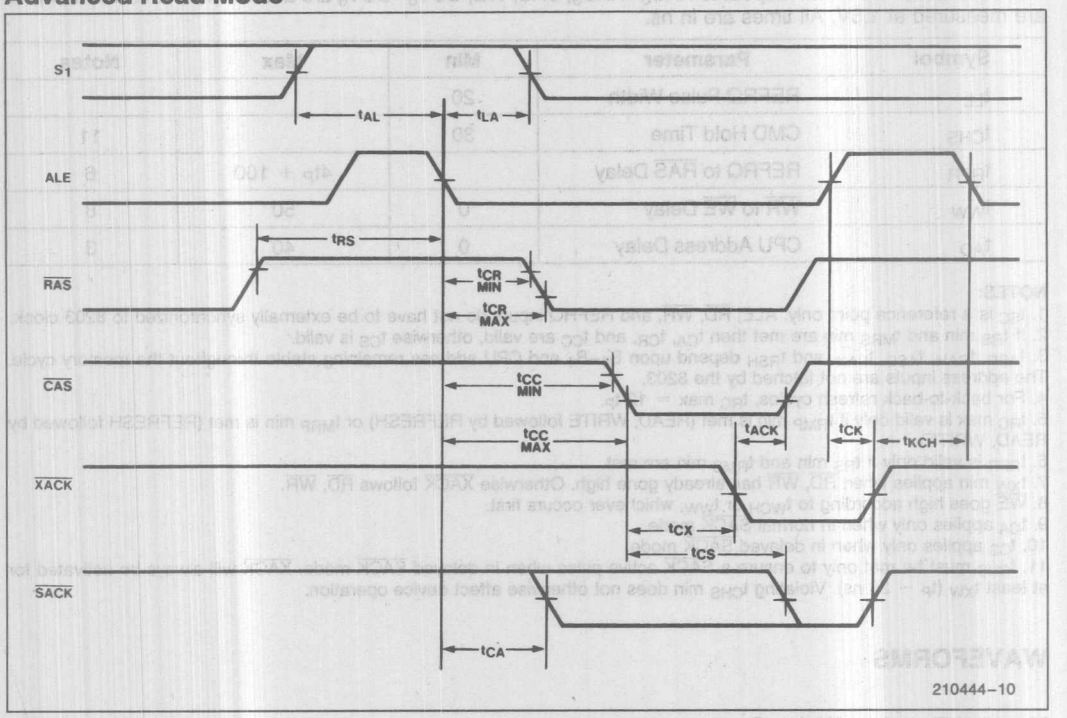
WAVEFORMS

Normal Read or Write Cycle

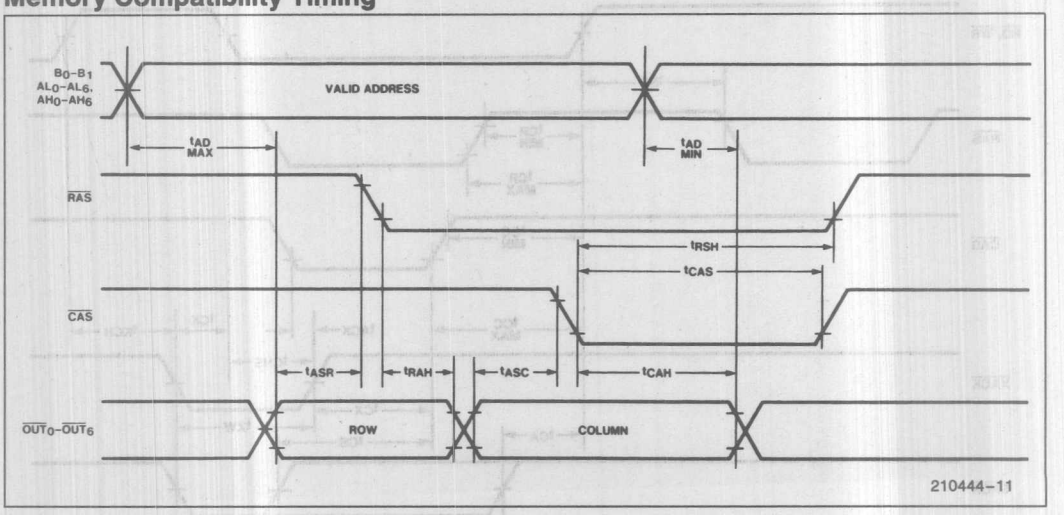


WAVEFORMS (Continued)

Advanced Read Mode

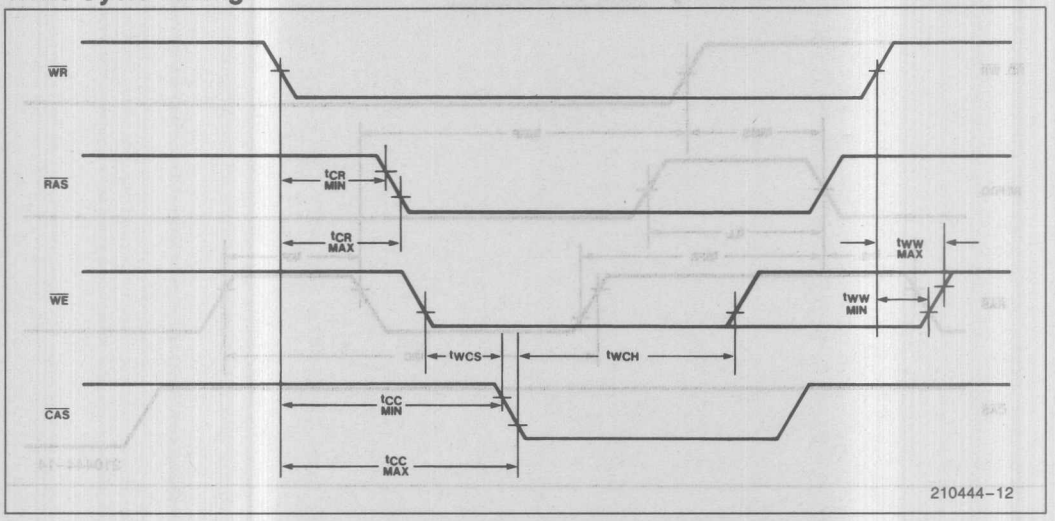


Memory Compatibility Timing

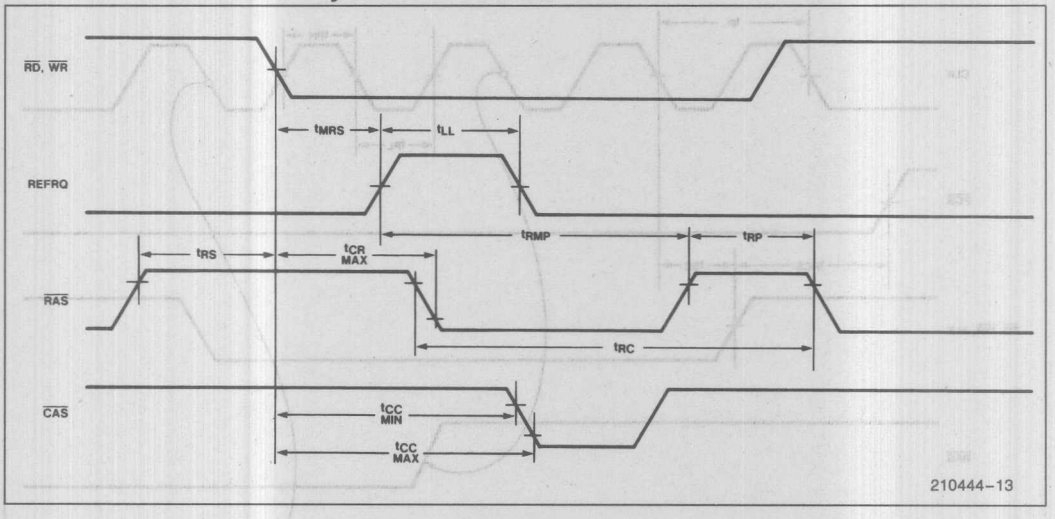


WAVEFORMS (Continued)

Write Cycle Timing

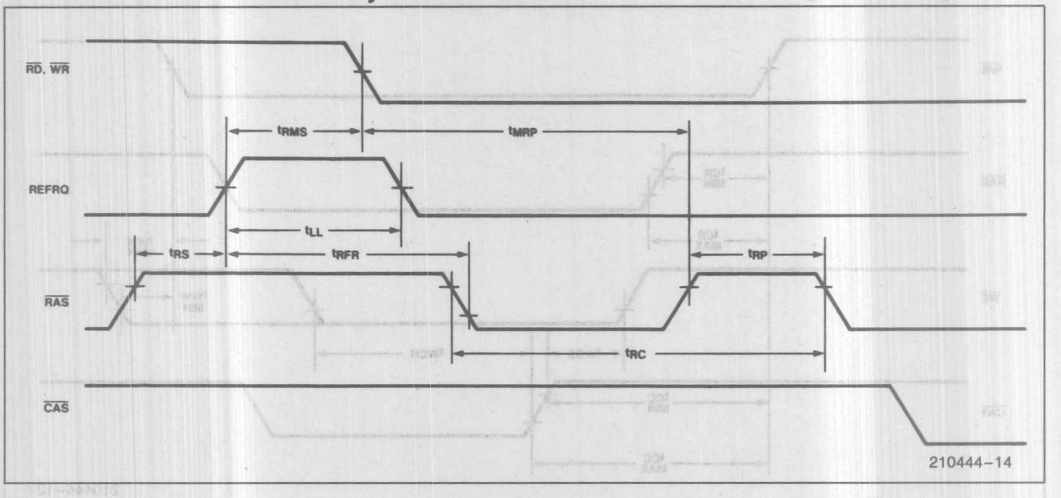


Read or Write Followed By External Refresh

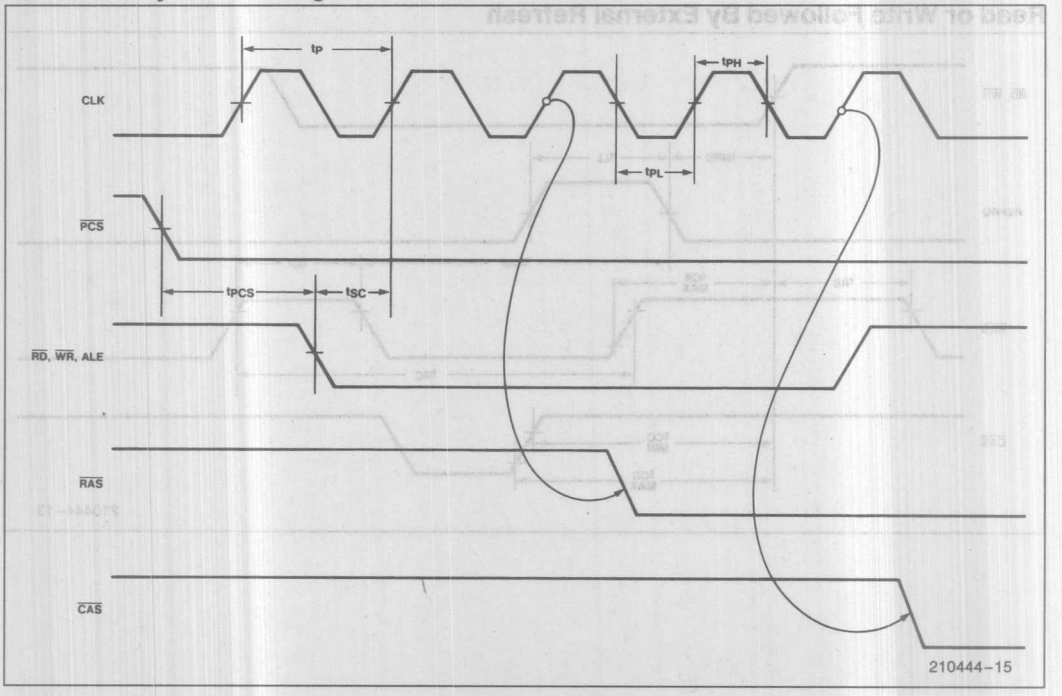


WAVEFORMS (Continued)

External Refresh Followed By Read or Write



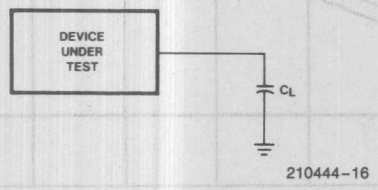
Clock and System Timing



are for the Test Load unless otherwise noted.

Pin	Test Load
$\overline{SACK}$, $\overline{XACK}$	C_L = 30 pF
$\overline{OUT_0}$–$\overline{OUT_6}$	C_L = 160 pF
$\overline{RAS_0}$–$\overline{RAS_3}$	C_L = 60 pF
$\overline{WE}$	C_L = 224 pF
$\overline{CAS}$	C_L = 320 pF

A.C. TESTING LOAD CIRCUIT



210444-16

NOTE:
C_L includes jig capacitance.

used to determine the effects of capacitive loading on the A.C. Timing Parameters. Using this design tool in conjunction with the timing waveforms, the designer can determine typical timing shifts based on system capacitive load.

Example: Find the effect on t_{CR} and t_{CC} using 32 64K Dynamic RAMs configured in 2 banks.

1) Determine the typical $\overline{RAS}$ and $\overline{CAS}$ capacitance:

   From the data sheet $\overline{RAS}$ = 5 pF and $\overline{CAS}$ = 5 pF.

   ∴ $\overline{RAS}$ load = 80 pF + board capacitance.

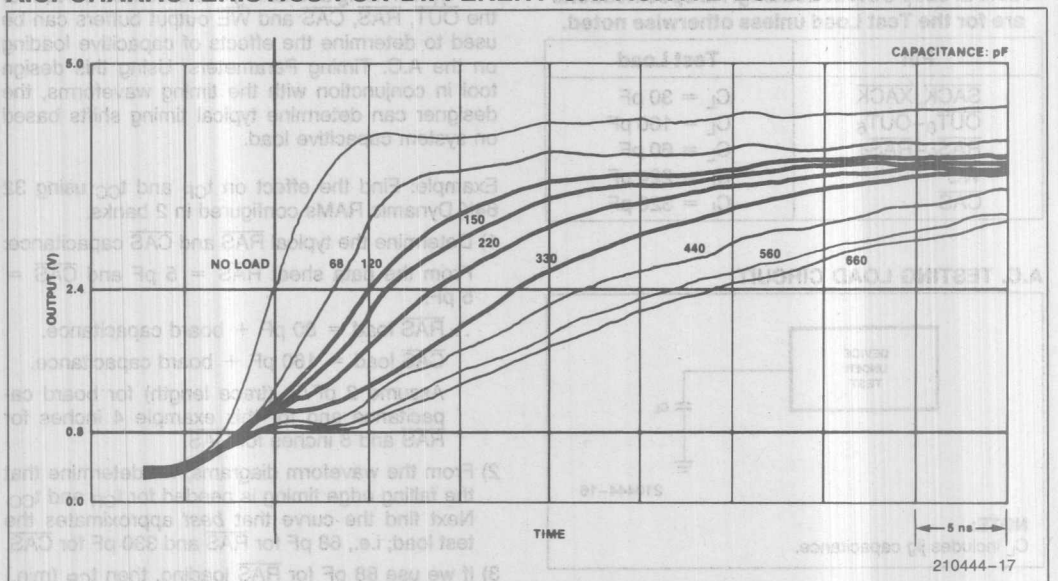
   $\overline{CAS}$ load = 160 pF + board capacitance.

   Assume 2 pF/in (trace length) for board capacitance and for this example 4 inches for $\overline{RAS}$ and 8 inches for $\overline{CAS}$.

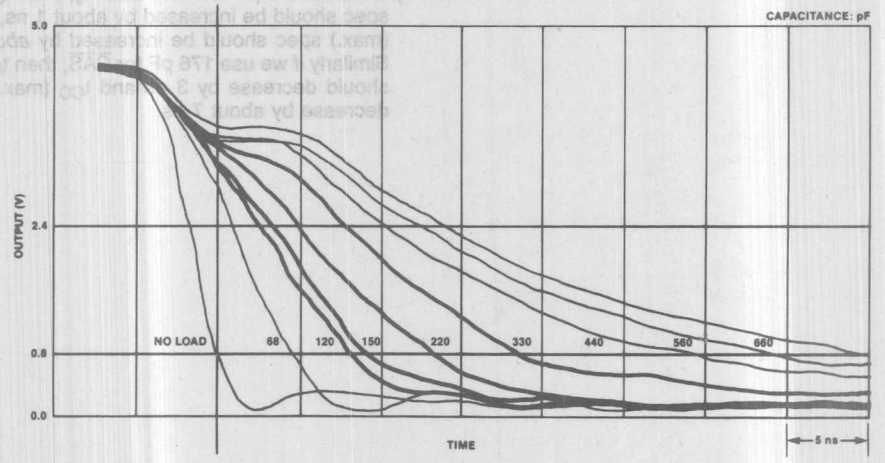
2) From the waveform diagrams, we determine that the falling edge timing is needed for t_{CR} and t_{CC}. Next find the curve that *best* approximates the test load; i.e., 68 pF for $\overline{RAS}$ and 330 pF for $\overline{CAS}$.

3) If we use 88 pF for $\overline{RAS}$ loading, then t_{CR} (min.) spec should be increased by about 1 ns, and t_{CR} (max.) spec should be increased by *about* 2 ns. Similarly if we use 176 pF for $\overline{CAS}$, then t_{CC} (min.) should decrease by 3 ns and t_{CC} (max.) should decrease by about 7 ns.

A.C. CHARACTERISTICS FOR DIFFERENT CAPACITIVE LOADS



210444-17



210444-18

NOTE:
Use the Test Load as the base capacitance for estimating timing shifts for system critical timing parameters.

MEASUREMENT CONDITIONS

Pins not measured are loaded with the Test Load capacitance

$T_A = 25°C$ $V_{CC} = +5V$ $t_P = 50$ ns

8206
ERROR DETECTION AND CORRECTION UNIT

- **Detects All Single Bit, and Double Bit and Most Multiple Bit Errors**
- **Corrects All Single Bit Errors**
- **3 Selections**

3 Selections	8206-1	8206
Detection	35 ns	42 ns
Correction	55 ns	67 ns

- **Syndrome Outputs for Error Logging**
- **Automatic Error Scrubbing with 8207**
- **Expandable to Handle 80 Bit Memories**
- **Separate Input and Output Busses—No Timing Strobes Required**
- **Supports Read With and Without Correction, Writes, Partial (Byte) Writes, and Read-Modify-Writes**
- **HMOS III Technology for Low Power**
- **68 Pin Leadless JEDEC Package**
- **68 Pin Grid Array Package**

The HMOS 8206 Error Detection and Correction Unit is a high-speed device that provides error detection and correction for memory systems (static and dynamic) requiring high reliability and performance. Each 8206 handles 8 or 16 data bits and up to 8 check bits. 8206's can be cascaded to provide correction and detection for up to 80 bits of data. Other 8206 features include the ability to handle byte writes, memory initialization, and error logging.

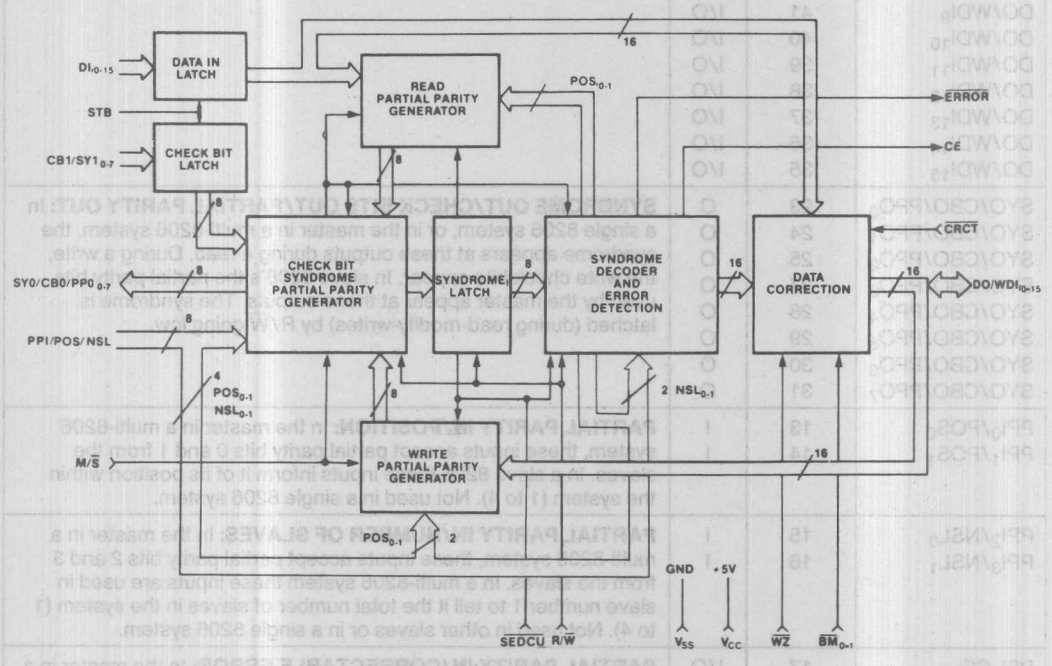


Figure 1. 8206 Block Diagram

September 1987
Order Number: 205220-008

Table 1. 8206 Pin Description

Symbol	Pin No.	Type	Name and Function
DI_{0-15}	1, 68-61, 59-53	I	**DATA IN:** These inputs accept a 16 bit data word from RAM for error detection and/or correction.
CBI/SYI_0	5	I	**CHECK BITS IN/SYNDROME IN:** In a single 8206 system,or in the master in a multi-8206 system, these inputs accept the check bits (5 to 8) from the RAM. In a single 8206 16 bit system, CBI_{0-5} are used. In slave 8206's these inputs accept the syndrome from the master.
CBI/SYI_1	6	I	
CBI/SYI_2	7	I	
CBI/SYI_3	8	I	
CBI/SYI_4	9	I	
CBI/SYI_5	10	I	
CBI/SYI_6	11	I	
CBI/SYI_7	12	I	
DO/WDI_0	51	I/O	**DATA OUT/WRITE DATA IN:** In a read cycle, data accepted by DI_{0-15} appears at these outputs corrected if $\overline{CRCT}$ is low, or uncorrected if $\overline{CRCT}$ is high. The $\overline{BM}$ inputs must be high to enable the output buffers during the read cycle. In a write cycle, data to be written into the RAM is accepted by these inputs for computing the write check bits. In a partial-write cycle, the byte not to be modified appears at either DO_{0-7} if $\overline{BM}_0$ is high, or DO_{8-15} if $\overline{BM}_1$ is high, for writing to the RAM. When $\overline{WZ}$ is active, it causes the 8206 to output all zeros at DO_{0-15}, with the proper write check bits on CBO.
DO/WDI_1	50	I/O	
DO/WDI_2	49	I/O	
DO/WDI_3	48	I/O	
DO/WDI_4	47	I/O	
DO/WDI_5	46	I/O	
DO/WDI_6	45	I/O	
DO/WDI_7	44	I/O	
DO/WDI_8	42	I/O	
DO/WDI_9	41	I/O	
DO/WDI_{10}	40	I/O	
DO/WDI_{11}	39	I/O	
DO/WDI_{12}	38	I/O	
DO/WDI_{13}	37	I/O	
DO/WDI_{14}	36	I/O	
DO/WDI_{15}	35	I/O	
$SYO/CBO/PPO_0$	23	O	**SYNDROME OUT/CHECK BITS OUT/PARTIAL PARITY OUT:** In a single 8206 system, or in the master in a multi-8206 system, the syndrome appears at these outputs during a read. During a write, the write check bits appear. In slave 8206's the partial parity bits used by the master appear at these outputs. The syndrome is latched (during read-modify-writes) by R/$\overline{W}$ going low.
$SYO/CBO/PPO_1$	24	O	
$SYO/CBO/PPO_2$	25	O	
$SYO/CBO/PPO_3$	27	O	
$SYO/CBO/PPO_4$	28	O	
$SYO/CBO/PPO_5$	29	O	
$SYO/CBO/PPO_6$	30	O	
$SYO/CBO/PPO_7$	31	O	
PPI_0/POS_0	13	I	**PARTIAL PARITY IN/POSITION:** In the master in a multi-8206 system, these inputs accept partial parity bits 0 and 1 from the slaves. In a slave 8206 these inputs inform it of its position within the system (1 to 4). Not used in a single 8206 system.
PPI_1/POS_1	14	I	
PPI_2/NSL_0	15	I	**PARTIAL PARITY IN/NUMBER OF SLAVES:** In the master in a multi-8206 system, these inputs accept partial parity bits 2 and 3 from the slaves. In a multi-8206 system these inputs are used in slave number 1 to tell it the total number of slaves in the system (1 to 4). Not used in other slaves or in a single 8206 system.
PPI_3/NSL_1	16	I	
PPI_4CE	17	I/O	**PARTIAL PARITY IN/CORRECTABLE ERROR:** In the master in a multi-8206 system this pin accepts partial parity bit 4. In slave number 1 only, or in a single 8206 system, this pin outputs the correctable error flag. CE is latched by R/$\overline{W}$ going low. Not used in other slaves.

Table 1. 8206 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
PPI_5 PPI_6 PPI_7	18 19 20	I I I	**PARTIAL PARITY IN:** In the master in a multi-8206 system these pins accept partial parity bits 5 to 7. The number of partial parity bits equals the number of check bits. Not used in single 8206 systems or in slaves.
$\overline{ERROR}$	22	O	**ERROR:** This pin outputs the error flag in a single 8206 system or in the master of a multi-8206 system. It is latched by R/$\overline{W}$ going low. Not used in slaves.
$\overline{CRCT}$	52	I	**CORRECT:** When low this pin causes data correction during a read or read-modify-write cycle. When high, it causes error correction to be disabled, although error checking is still enabled.
$\overline{STB}$	2	I	**STROBE:** STB is an input control used to strobe data at the DI inputs and check-bits at the CBI/SYI inputs. The signal is active high to admit the inputs. The signals are latched by the high-to-low transition of STB.
$\overline{BM}_0$ $\overline{BM}_1$	33 32	I I	**BYTE MARKS:** When high, the Data Out pins are enabled for a read cycle. When low, the Data Out buffers are tristated for a write cycle. $\overline{BM}_0$ controls DO_{0-7}, while $\overline{BM}_1$ controls DO_{8-15}. In partial (byte) writes, the byte mark input is low for the new byte to be written.
R/$\overline{W}$	21	I	**READ/WRITE:** When high this pin causes the 8206 to perform detection and correction (if $\overline{CRCT}$ is low). When low, it causes the 8206 to generate check-bits. On the high-to-low transition the syndrome is latched internally for read-modify-write cycles.
$\overline{WZ}$	34	I	**WRITE ZERO:** When low this input overrides the $\overline{BM}_{0-1}$ and R/$\overline{W}$ inputs to cause the 8206 to output all zeros at DO_{0-15} with the corresponding check-bits at CBO_{0-7}. Used for memory initialization.
M/$\overline{S}$	4	I	**MASTER/SLAVE:** Input tells the 8206 whether it is a master (high) or a slave (low).
$\overline{SEDCU}$	3	I	**SINGLE EDC UNIT:** Input tells the master whether it is operating as a single 8206 (low) or as the master in a multi-8206 system (high). Not used in slaves.
V_{CC}	60	I	**POWER SUPPLY:** +5V
V_{SS}	26	I	**LOGIC GROUND**
V_{SS}	43	I	**OUTPUT DRIVER GROUND**

FUNCTIONAL DESCRIPTION

The 8206 Error Detection and Correction Unit provides greater memory system reliability through its ability to detect and correct memory errors. It is a single chip device that can detect and correct all single bit errors and detect all double bit and some higher multiple bit errors. Some other odd multiple bit errors (e.g., 5 bits in error) are interpreted as single bit errors, and the CE flag is raised. While some even multiple bit errors (e.g., 4 bits in error) are interpreted as no error, most are detected as double bit errors. This error handling is a function of the number of check bits used by the 8206 (see Figure 2) and the specific Hamming code used. Errors in check bits are not distinguished from errors in a word.

For more information on error correction codes, see Intel Application Notes AP-46 and AP-73.

A single 8206 handles 8 or 16 bits of data, and up to 5 8206's can be cascaded in order to handle data paths of 80 bits. For a single 8206 8 bit system, the DI_{8-15}, DO/WDI_{8-15} and $\overline{BM}_1$ inputs are grounded. See the Multi-Chip systems section for information on 24-80 bit systems.

The 8206 has a "flow through" architecture. It supports two kinds of error correction architecture: 1) Flow-through, or correct-always; and 2) Parallel, or check-only. These are two separate 16-pin busses,

Data Word Bits	Check Bits
8	5
16	6
24	6
32	7
40	7
48	8
56	8
64	8
72	8
80	8

Figure 3. Number of Check Bits Used by 8206

one to accept data from the RAM (DI) and the other to deliver corrected data to the system bus (DO/WDI). The logic is entirely combinatorial during a read cycle. This is in contrast to an architecture with only one bus, with bidirectional bus drivers that must first read the data and then be turned around to output the corrected data. The latter architecture typically requires additional hardware (latches and/or transceivers) and may be slower in a system due to timing skews of control signals.

READ CYCLE

With the R/$\overline{W}$ pin high, data is received from the RAM outputs into the DI pins where it is optionally latched by the STB signal. Check bits are generated from the data bits and compared to the check bits read from the RAM into the CBI pins. If an error is detected the $\overline{ERROR}$ flag is activated and the correctable error flag (CE) is used to inform the system whether the error was correctable or not. With the $\overline{BM}$ inputs high, the word appears corrected at the DO pins if the error was correctable, or unmodified if the error was uncorrectable.

If more than one 8206 is being used, then the check bits are read by the master. The slaves generate a partial parity output (PPO) and pass it to the master. The master 8206 then generates and returns the syndrome to the slaves (SYO) for correction of the data.

The 8206 may alternatively be used in a "check-only" mode with the $\overline{CRCT}$ pin left high. With the correction facility turned off, the propagation delay from memory outputs to 8206 outputs is significantly shortened. In this mode the 8206 issues an $\overline{ERROR}$ flag to the CPU, which can then perform one of several options: lengthen the current cycle for correction, restart the instruction, perform a diagnostic routine, etc.

A syndrome word, five to eight bits in length and containing all necessary information about the existence and location of an error, is made available to the system at the SYO_{0-7} pins. Error logging may be accomplished by latching the syndrome and the memory address of the word in error.

WRITE CYCLE

For a full write, in which an entire word is written to memory, the data is written directly to the RAM, bypassing the 8206. The same data enters the 8206 through the WDI pins where check bits are generated. The Byte Mark inputs must be low to tristate the DO drivers. The check bits, 5 to 8 in number, are then written to the RAM through the CBO pins for storage along with the data word. In a multi-chip system, the master writes the check bits using partial parity information from the slaves.

In a partial write, part of the data word is overwritten, and part is retained in memory. This is accomplished by performing a read-modify-write cycle. The complete old word is read into the 8206 and corrected, with the syndrome internally latched by R/$\overline{W}$ going low. Only that part of the word not to be modified is output onto the DO pins, as controlled by the Byte Mark inputs. That portion of the word to be overwrit-

ten is supplied by the system bus. The 8206 then calculates check bits for the new word, using the byte from the previous read and the new byte from the system bus, and writes them to the memory.

READ-MODIFY-WRITE CYCLES

Upon detection of an error the 8206 may be used to correct the bit in error in memory. This reduces the probability of getting multiple-bit errors in subsequent read cycles. This correction is handled by executing read-modify-write cycles.

The read-modify-write cycle is controlled by the R/$\overline{W}$ input. After (during) the read cycle, the system dynamic RAM controller or CPU examines the 8206 $\overline{ERROR}$ and CE outputs to determine if a correctable error occurred. If it did, the dynamic RAM controller or CPU forces R/$\overline{W}$ low, telling the 8206 to latch the generated syndrome and drive the corrected check bits onto the CBO outputs. The corrected data is available on the DO pins. The DRAM controller then writes the corrected data and corresponding check bits into memory.

The 8206 may be used to perform read-modify-writes in one or two RAM cycles. If it is done in two cycles, the 8206 latches are used to hold the data and check bits from the read cycle to be used in the following write cycle. The Intel 8207 Dual Port Dynamic RAM controller allows read-modify-write cycles in one memory cycle. See the System Environment section.

INITIALIZATION

A memory system operating with ECC requires some form of initialization at system power-up in order to set valid data and check bit information in memory. The 8206 supports memory initialization by the write zero function. By activating the $\overline{WZ}$ pin, the 8206 will write a data pattern of zeros and the associated check bits in the current write cycle. By thus writing to all memory at power-up, a controller can set memory to valid data and check bits. Massive memory failure, as signified by both data and check bits all ones or zeros, will be detected as an uncorrectable error.

MULTI-CHIP SYSTEMS

A single 8206 handles 8 or 16 bits of data and 5 or 6 check bits, respectively. Up to 5 8206's can be cascaded for 80 bit memories with 8 check bits.

When cascaded, one 8206 operates as a master, and all others as slaves. As an example, during a read cycle in a 32 bit system with one master and one slave, the slave calculates parity on its portion of the word—"partial parity"—and presents it to the master through the PPO pins. The master combines the partial parity from the slave with the parity it calculated from its own portion of the word to generate the syndrome. The syndrome is then returned by the master to the slave for error correction. In systems with more than one slave the above description continues to apply, except that the partial parity outputs of the slaves must be XOR'd externally. Figure 4 shows the necessary external logic for multi-chip systems. Write and read-modify-write cycles are carried out analogously. See the System Operation section for multi-chip wiring diagrams.

There are several pins used to define whether the 8206 will operate as a master or a slave. Tables 3 and 4 illustrate how these pins are tied.

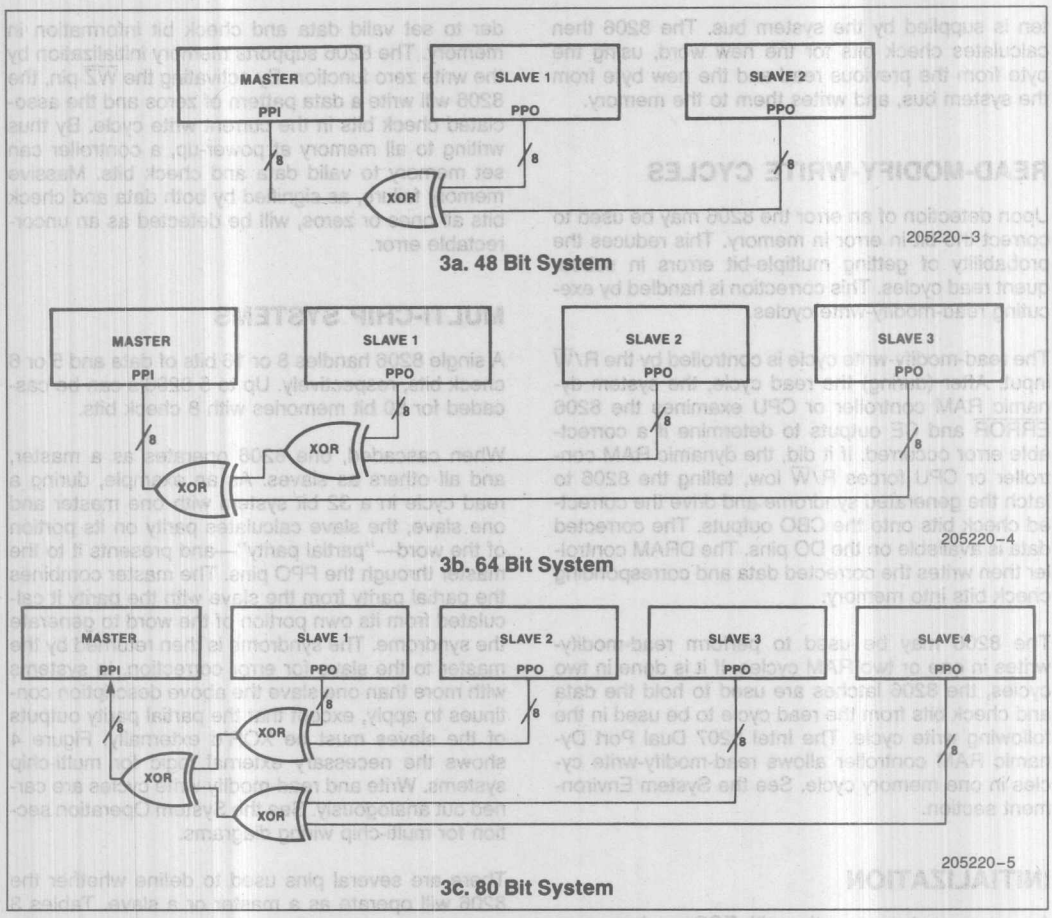


Figure 4. External Logic for Multi-Chip Sytems

Table 3. Master/Slave Pin Assignments

Pin No.	Pin Name	Master	Slave 1	Slave 2	Slave 3	Slave 4
4	$M/\overline{S}$	+5V	gnd	gnd	gnd	gnd
3	$\overline{SEDCU}$	+5V	+5V	+5V	+5V	+5V
13	PPI_0/POS_0	PPI	gnd	+5V	gnd	+5V
14	PPI_1/POS_1	PPI	gnd	gnd	+5V	+5V
15	PPI_2/NSL_0	PPI	*	+5V	+5V	+5V
16	PPI_3/NSL_1	PPI	*	+5V	+5V	+5V

NOTE:
Pins 13, 14, 15, 16 have internal pull-up resistors and may be left as N.C. where specified as connecting to +5V.

Table 4. NSL Pin Assignments for Slave 1

	Number of Slaves			
Pin	**1**	**2**	**3**	**4**
PPI_2/NSL_0	GND	+5V	GND	+5V
PPI_3/NSL_1	GND	GND	+5V	+5V

The timing specifications for multi-chip systems must be calculated to take account of the external XOR gating in 3, 4 and 5-chip systems. Let tXOR be the delay for a single external TTL XOR gate. Then the following equations show how to calculate the relevant timing parameters for 2-chip (n = 0), 3-chip (n = 1), 4-chip (n = 2), and 5-chip (n = 2) systems:

Data-in to corrected data-out (read cycle) =

$$TDVSV + TPVSV + TSVQV + ntXOR$$

Data-in to error flag (read cycle) =

$$TDVSV + TPVEV + ntXOR$$

Data-in to correctable error flag (read cycle) =

$$TDVSV + TPVSV + TSVCV + ntXOR$$

Write data to check-bits valid (full write cycle) =

$$TQVQV + TPVSV + ntXOR$$

Data-in to check-bits valid (read-mod-write cycle) =

$$TDVSV + TPVSV + TSVQV + TQVQV + TPVSV + 2ntXOR$$

Data-in to check-bits valid (non-correcting read-modify-write cycle) =

$$TDVQU + TQVQV + TPVSV + ntXOR$$

HAMMING CODE

The 8206 uses a modified Hamming code which was optimized for multi-chip EDCU systems. The code is such that partial parity is computed by all 8206's in parallel. No 8206 requires more time for propagation through logic levels than any other one, and hence no one device becomes a bottleneck in the parity operation. However, one or two levels of external TTL XOR gates are required in systems with three to five chips. The code appears in Table 5. The check bits are derived from the table by XORing or XNORing together the bits indicated by 'X's in each row corresponding to a check bit. For example, check bit 0 in the MASTER for data word 1000110101101011 will be "0". It should be noted that the 8206 will detect the gross-error condition of all lows or all highs.

Error correction is accomplished by identifying the bad bit and inverting it. Table 5 can also be used as an error syndrome table by replacing the 'X's with '1's. Each column then represents a different syndrome word, and by locating the column corresponding to a particular syndrome the bit to be corrected may be identified. If the syndrome cannot be located then the error cannot be corrected. For example, if the syndrome word is 00110111, the bit to be corrected is bit 5 in the slave one data word (bit 21).

The syndrome decoding is also summarized in Tables 6 and 7 which can be used for error logging. By finding the appropriate syndrome word (starting with bit zero, the least significant bit), the result is either: 1) no error; 2) an identified (correctable) single bit error; 3) a double bit error; or 4) a multi-bit uncorrectable error.

Table 5. Modified Hamming Code Check Bit Generation

Check bits are generated by XOR'ing (except for the CB0 and CB1 data bits, which are XNOR'ed in the Master) the data bits in the rows corresponding to the check bits. Note there are 6 check bits in a 16-bit system, 7 in a 32-bit system, and 8 in 48-or-more-bit systems.

BYTE NUMBER		0								1								OPERATION
BIT NUMBER		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CHECK BITS	CB0 =	x	x	-	x	-	x	x	-	x	-	-	x	-	x	-	-	XNOR
	CB1 =	x	-	x	-	-	x	-	x	-	x	-	x	x	-	x	-	XNOR
	CB2 =	-	x	x	-	x	-	x	x	-	-	x	-	x	-	-	x	XOR
	CB3 =	x	x	x	x	x	-	-	-	x	x	x	-	-	-	-	-	XOR
	CB4 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	XOR
	CB5 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
	CB6 =	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
	CB7 =	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
DATA BITS		0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	
		0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	
		16 BIT OR MASTER																

BYTE NUMBER		2								3								OPERATION
BIT NUMBER		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CHECK BITS	CB0 =	-	x	x	x	-	x	x	-	-	x	x	-	-	x	-	-	XOR
	CB1 =	x	x	x	-	-	x	-	x	x	x	-	-	-	-	-	x	XOR
	CB2 =	-	x	x	x	-	x	x	x	-	-	x	x	-	-	-	-	XOR
	CB3 =	x	x	-	-	x	-	x	x	x	-	-	x	x	-	-	-	XOR
	CB4 =	x	x	-	-	x	x	x	x	-	-	-	-	x	-	x	-	XOR
	CB5 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	XOR
	CB6 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
	CB7 =	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
DATA BITS		1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3	
		6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	
		SLAVE #1																

BYTE NUMBER		4								5								6								7								8								9								OPERATION
BIT NUMBER		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CHECK BITS	CB0 =	x	x	-	x	-	x	x	-	x	-	-	x	-	x	-	-	x	-	x	x	-	-	x	-	x	-	x	x	-	-	x	-	-	x	x	-	x	x	-	-	-	x	x	-	-	x	-	-	XOR
	CB1 =	x	-	x	-	-	x	-	x	-	x	-	x	x	-	x	-	-	x	x	-	-	-	x	x	x	x	x	-	-	-	x	-	-	x	x	x	-	x	x	-	-	-	x	x	-	-	-	-	XOR
	CB2 =	-	x	x	-	x	-	x	x	-	-	x	-	x	-	-	x	-	x	x	x	-	x	x	-	-	x	x	-	-	x	-	-	x	-	-	x	-	x	x	-	x	-	x	x	-	-	x	-	x	XOR
	CB3 =	x	x	x	x	x	-	-	-	x	x	x	-	-	-	-	-	x	-	x	-	-	x	x	-	x	x	-	-	x	x	-	-	-	x	x	x	x	-	-	x	x	x	-	-	-	x	-	-	-	XOR
	CB4 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	-	x	x	-	-	-	x	x	x	x	-	-	-	x	-	XOR
	CB5 =	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x	-	x	x	x	x	-	x	-	-	-	x	-	-	-	x	XOR
	CB6 =	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	-	-	x	x	x	x	-	-	-	-	-	x	-	x	XOR
	CB7 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
DATA BITS		3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	5	6	6	6	6	6	6	6	7	7	7	7	7	7	7	7	7	7	7	7	7	
		2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	
		SLAVE #2								SLAVE #3								SLAVE #4																																

Table 6. 8206 Syndrome Decoding

Syndrome Bits 7 6 5 4	0 0 1 0 1 0 1 0 1 0 1 0 1 0 1	0 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1	0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 1	0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0	N	CB0	CB1	D	CB2	D	D	18	CB3	D	D	0	D	1	2	D
0 0 0 1	CB4	D	D	5	D	6	7	D	D	D	16	D	4	D	D	17
0 0 1 0	CB5	D	D	11	D	19	12	D	D	8	9	D	10	D	D	67
0 0 1 1	D	13	14	D	15	D	D	21	20	D	D	66	D	22	23	D
0 1 0 0	CB6	D	D	25	D	26	49	D	D	48	24	D	27	D	D	50
0 1 0 1	D	52	55	D	51	D	D	70	28	D	D	65	D	53	54	D
0 1 1 0	D	29	31	D	64	D	D	69	68	D	D	32	D	33	34	D
0 1 1 1	30	D	D	37	D	38	39	D	D	35	71	D	36	D	D	U
1 0 0 0	CB7	D	D	43	D	77	44	D	D	40	41	D	42	D	D	U
1 0 0 1	D	45	46	D	47	D	D	74	72	D	D	U	D	73	U	D
1 0 1 0	D	59	75	D	79	D	D	58	60	D	D	56	D	U	57	D
1 0 1 1	63	D	D	62	D	U	U	D	D	U	U	D	61	D	D	U
1 1 0 0	D	U	U	D	U	D	D	U	76	D	D	U	D	U	U	D
1 1 0 1	78	D	D	U	D	U	U	D	D	U	U	D	U	D	D	U
1 1 1 0	U	D	D	U	D	U	U	D	D	U	U	D	U	D	D	U
1 1 1 1	D	U	U	D	U	D	D	U	U	D	D	U	D	U	U	D

N = No Error
CBX = Error in Check Bit X
X = Error in Data Bit X
D = Double Bit Error
U = Uncorrectable Multi-Bit Error

SYSTEM ENVIRONMENT

The 8206 interface to a typical 32 bit memory system is illustrated in Figure 5. For larger systems, the partial parity bits from slaves two to four must be XOR'ed externally, which calls for one level of XOR gating for three 8206's and two levels for four or five 8206's.

The 8206 is designed for direct connection to the Intel 8207 Dynamic RAM Controller. The 8207 has the ability to perform dual port memory control, and Figure 6 illustrates a highly integrated dual port RAM implementation using the 8206 and 8207. The 8206/8207 combination permits such features as automatic scrubbing (correcting errors in memory during refresh), extending RAS and CAS timings for Read-Modify-Writes in single memory cycles, and automatic memory initialization upon reset. Together these two chips provide a complete dual-port, error-corrected dynamic RAM subsystem.

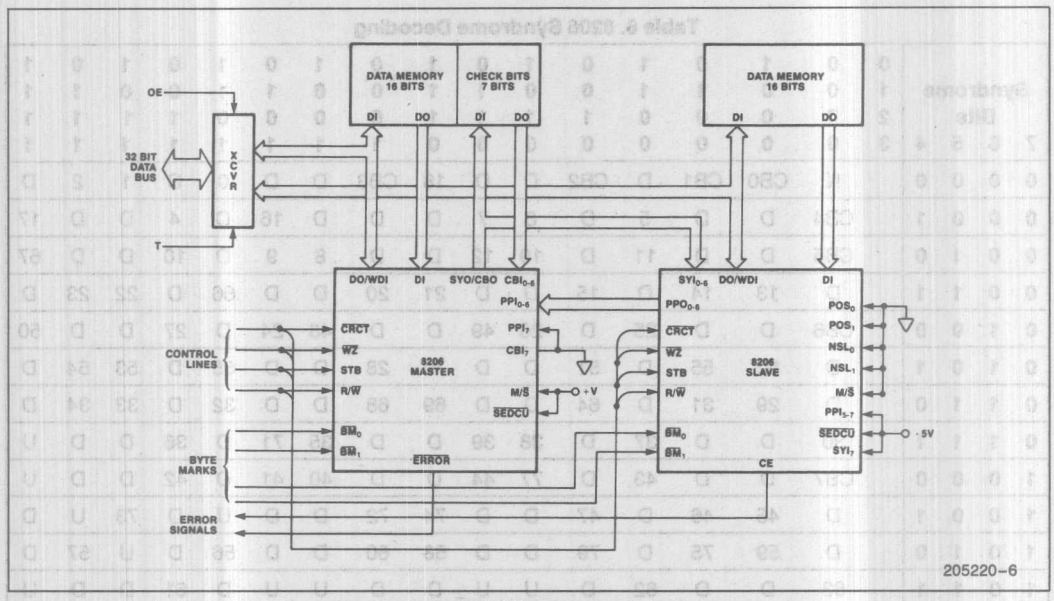


Figure 5. 32-Bit 8206 System Interface

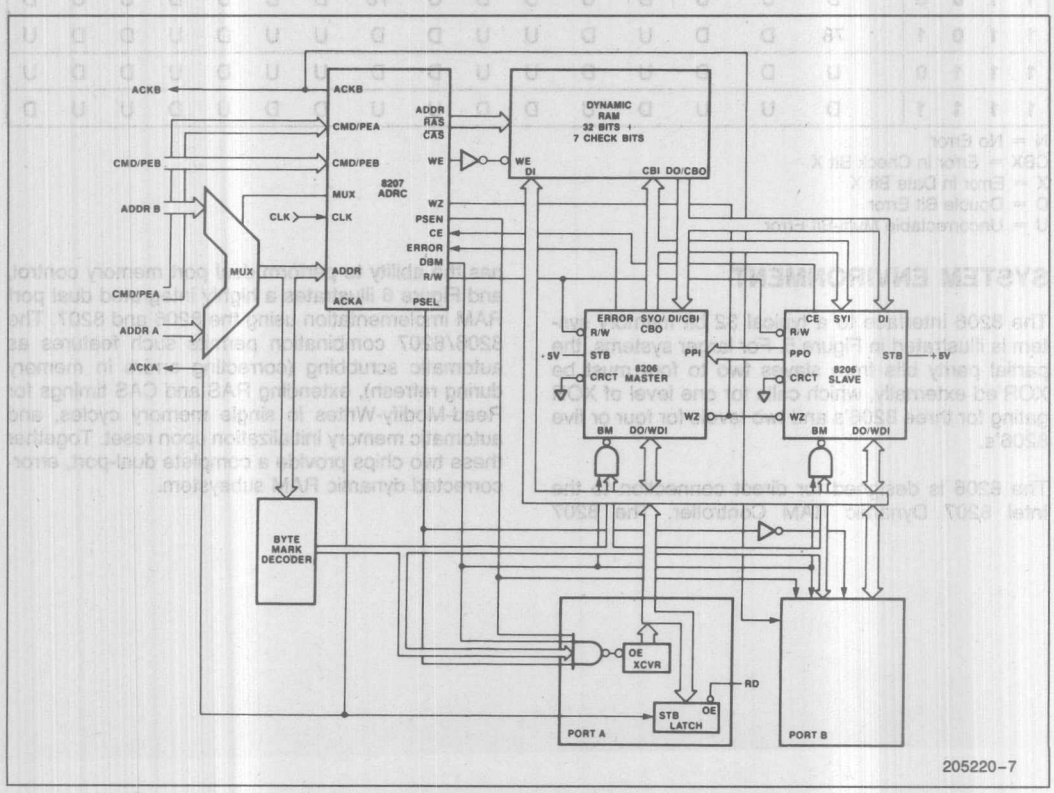


Figure 6. Dual Port RAM Subsystem with 8206/8207 (32-bit bus)

MEMORY BOARD TESTING

The 8206 lends itself to straightforward memory board testing with a minimum of hardware overhead. The following is a description of four common test modes and their implementation.

Mode 0— Read and write with error correction.
Implementation: This mode is the normal 8206 operating mode.

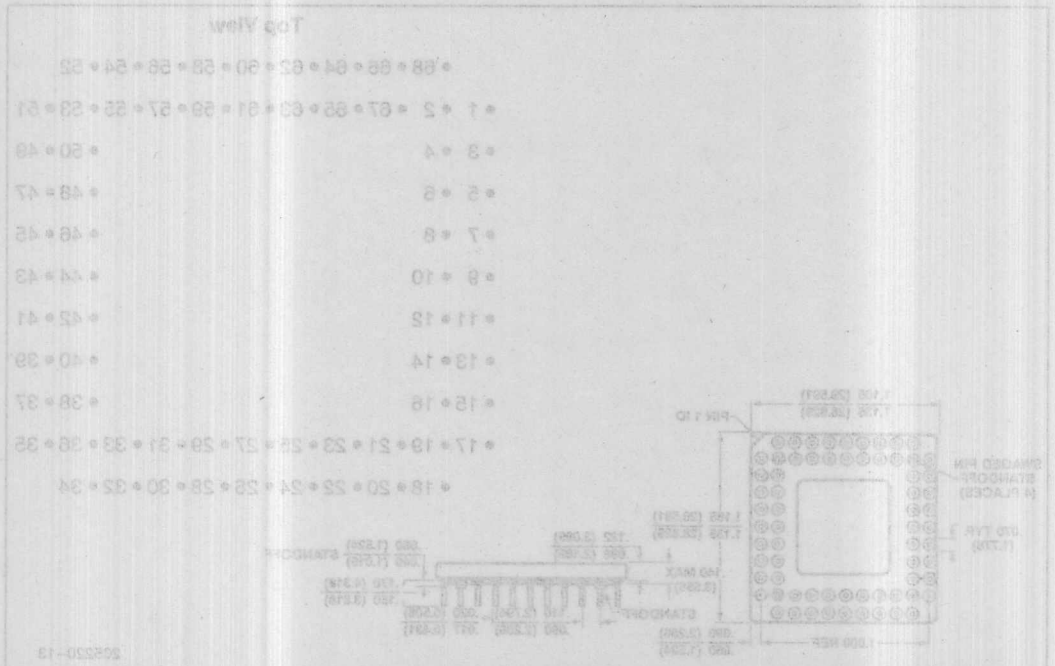
Mode 1— Read and write data with error correction disabled to allow test of data memory.
Implementation: This mode is performed with $\overline{CRCT}$ deactivated.

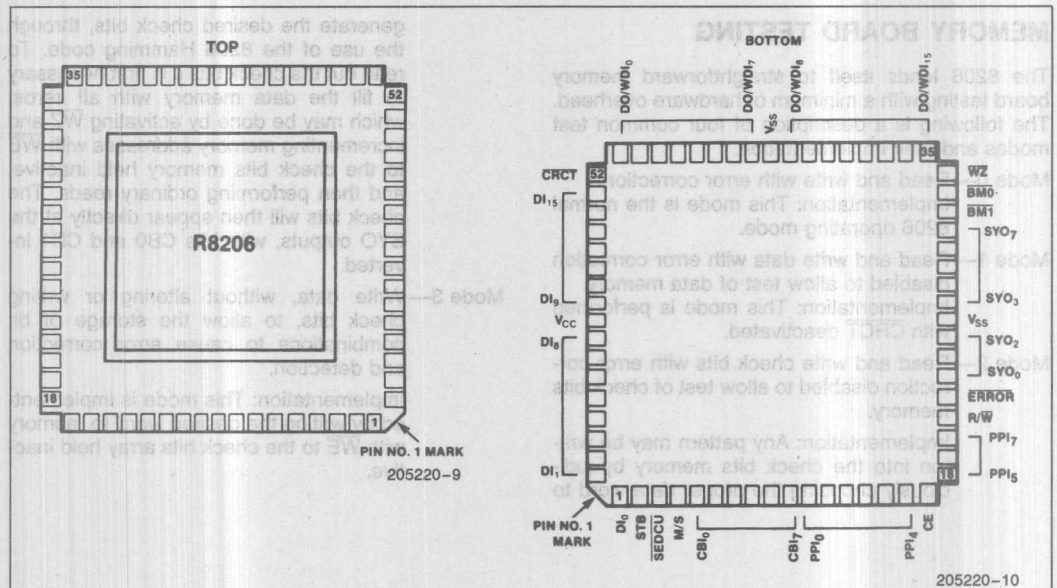
Mode 2— Read and write check bits with error correction disabled to allow test of check bits memory.

Implementation: Any pattern may be written into the check bits memory by judiciously choosing the proper data word to generate the desired check bits, through the use of the 8206 Hamming code. To read out the check bits it is first necessary to fill the data memory with all zeros, which may be done by activating $\overline{WZ}$ and incrementing memory addresses with $\overline{WE}$ to the check bits memory held inactive, and then performing ordinary reads. The check bits will then appear directly at the SYO outputs, with bits CB0 and CB1 inverted.

Mode 3— Write data, without altering or writing check bits, to allow the storage of bit combinations to cause error correction and detection.

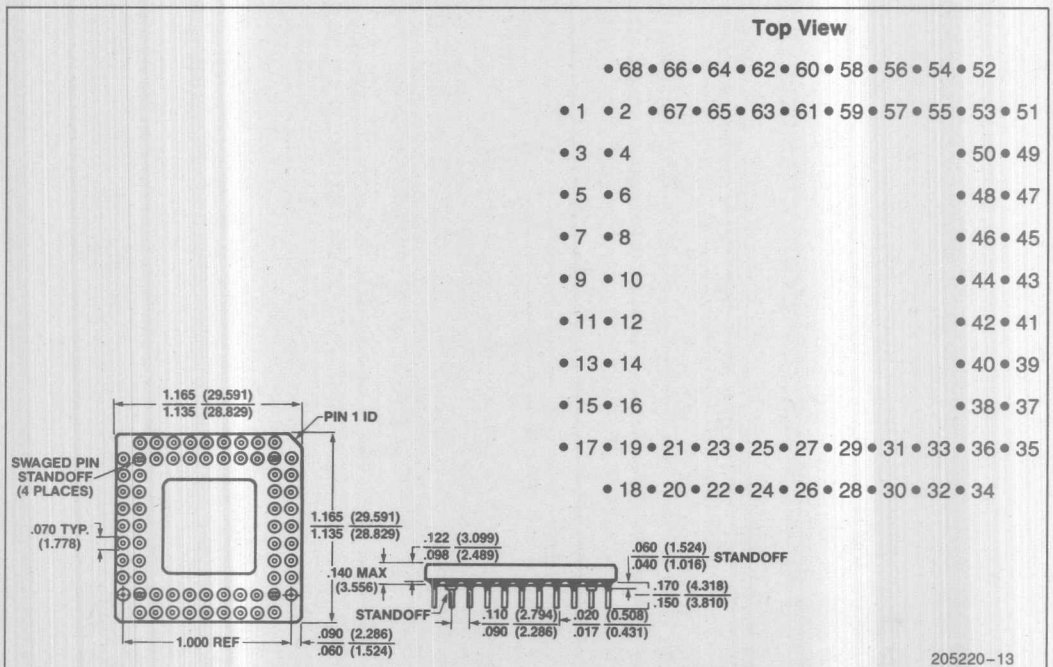
Implementation: This mode is implemented by writing the desired word to memory with $\overline{WE}$ to the check bits array held inactive.





NOTE:
68 pin JEDEC TYPE A hermetic chip carrier

Figure 8a. 8206 Leadless Chip Carrier (LCC) Pinout Diagram



NOTE:
68 lead ceramic pin grid array package, type A

Figure 8b. 8206 Pin Grid Array (PGA) Package and Pinout Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On Any Pin
with Respect to Ground.....−0.5V to +7V
Power Dissipation.....1.5W

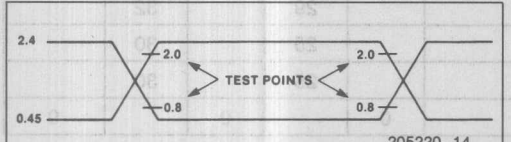
**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5.0V ± 10%, V_{SS} = GND

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CC}	Power Supply Current —Single 8206 or Slave #1 —Master in Multi-Chip or Slaves #2, 3, 4		270 230	mA mA	
V_{IL}(1)	Input Low Voltage	−0.5	0.8	V	
V_{IH}(1)	Input High Voltage	2.0	V_{CC} + 0.5V	V	
V_{OL}	Output Low Voltage —DO —All Others		0.45 0.45	V V	I_{OL} = 8 mA I_{OL} = 2.0 mA
V_{OH}	Output High Voltage —DO, CBO —All Other Outputs	2.6 2.4		V V	I_{OH} = −2 mA I_{OH} = −0.4 mA
I_{LO}	I/O Leakage Current —PPI_4/CE —DO/WDI_{0-15}		±20 ±10	μA μA	0.45V ≤ $V_{I/O}$ ≤ V_{CC}
I_{LI}	Input Leakage Current —$PPI_{0-3,5-7}$, CBI_{6-7}, $\overline{SEDCU}$(2) —All Other Input Only Pins		±20 ±10	μA μA	0V ≤ V_{IN} ≤ V_{CC}

NOTES:
1. $\overline{SEDCU}$ (pin 3) and M/$\overline{S}$ (pin 4) are device strapping options and should be tied to V_{CC} or GND. V_{IH} min = V_{CC} −0.5V and V_{IL} max = 0.5V.
2. PPI_{0-7} (pins 13–20) and CBI_{6-7} (pins 11, 12) have internal pull-up resistors and if left unconnected will be pulled to V_{CC}.

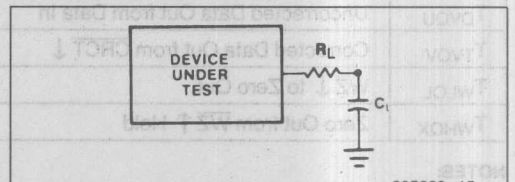
A.C. TESTING INPUT, OUTPUT WAVEFORM



205220-14

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT



205220-15

C_L Includes Jig Capacitance

A.C. CHARACTERISTICS

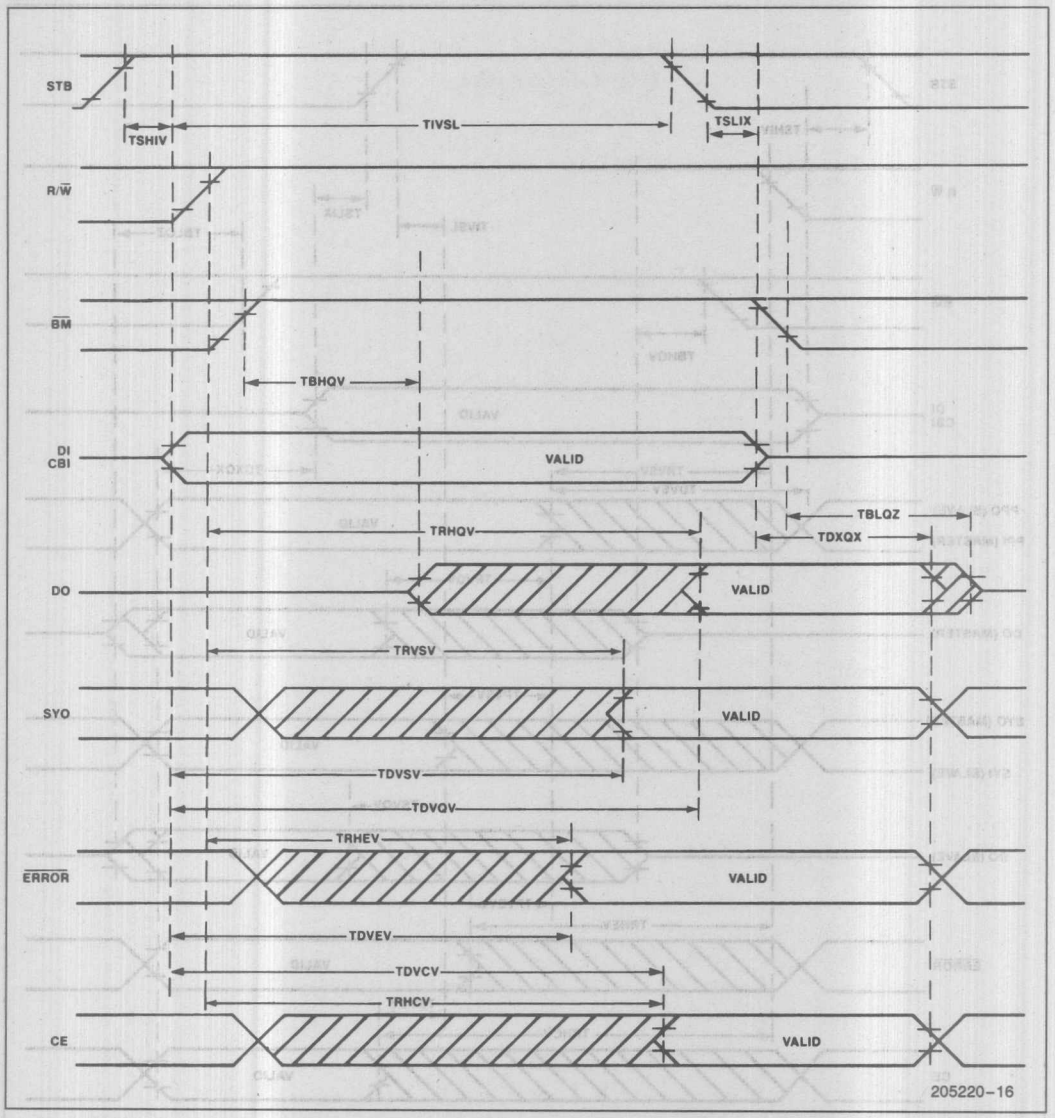
T_A = 0°C to 70°C, V_{CC} = +5V ± 10%, V_{SS} = 0V, R_L = 22Ω, C_L = 50 pF; all times are in ns

Symbol	Parameter	8206-1		8206		Notes
		Min	Max	Min	Max	
T_{RHEV}	$\overline{ERROR}$ Valid from R/$\overline{W}$ ↑		20		25	
T_{RHCV}	CE Valid from R/$\overline{W}$ ↑ (Single 8206)		34		44	
T_{RHQV}	Corrected Data Valid from R$\overline{W}$ ↑		44		54	1
T_{RVSV}	SYO/CBO/PPO Valid from R/$\overline{W}$		32		42	1
T_{DVEV}	$\overline{ERROR}$ Valid from Data/Check Bits In		35		42	
T_{DVCV}	CE Valid from Data/Check Bits In		50		70	
T_{DVQV}	Corrected Data Valid from Data/Check Bits In		55		67	
T_{DVSV}	SYO/PPO Valid from Data/Check Bits In		40		55	
T_{BHQV}	Corrected Data Access Time		35		37	
T_{BXQX}	Hold Time from Data/Check Bits In	0		0		1
T_{BLQZ}	Corrected Data Float Delay	0	25	0	28	1
T_{SHIV}	STB High to Data/Check Bits In Valid	30		30		2
T_{IVSL}	Data/Check Bits In to STB ↓ Set-Up	5		5		
T_{SLIX}	Data/Check Bits In from STB ↓ Hold	15		25		
T_{PVEV}	$\overline{ERROR}$ Valid from Partial Parity In		21		30	3
T_{PVQV}	Corrected Data (Master) from Partial Parity In		46		61	1, 3
T_{PVSV}	Syndrome/Check Bits Out from Partial Parity In		32		43	1, 3
T_{SVQV}	Corrected Data (Slave) Valid from Syndrome		41		51	3
T_{SVCV}	CE Valid from Syndrome (Slave Number 1)		43		48	3
T_{QVQV}	Check Bits/Partial Parity Out from Write Data In		44		64	1
T_{RHSX}	Check Bits/Partial Parity Out from R/W, $\overline{WZ}$ Hold	0		0		1
T_{RLSX}	Syndrome Out from R/$\overline{W}$ Hold	0		0		
T_{QXQX}	Hold Time from Write Data In	0		0		1
T_{SVRL}	Syndrome Out to R/$\overline{W}$ ↓ Set-Up	5		17		3
T_{DVRL}	Data/Check Bits to R/$\overline{W}$ Set-Up	24		39		1
T_{DVQU}	Uncorrected Data Out from Data In		29		32	
T_{TVQV}	Corrected Data Out from $\overline{CRCT}$ ↓		25		30	
T_{WLQL}	$\overline{WZ}$ ↓ to Zero Out		25		30	
T_{WHQX}	Zero Out from $\overline{WZ}$ ↑ Hold	0		0		0

NOTES:
1. A.C. Test Levels for CBO and DO are 2.4V and 0.8V.
2. T_{SHIV} is required to guarantee output delay timings: T_{DVEV}, T_{DVCV}, T_{DVQV}, T_{DVSV}, T_{SHIV} + T_{IVSL} guarantees a min STB pulse width of 35 ns.
3. Not required for 8/16 bit systems.

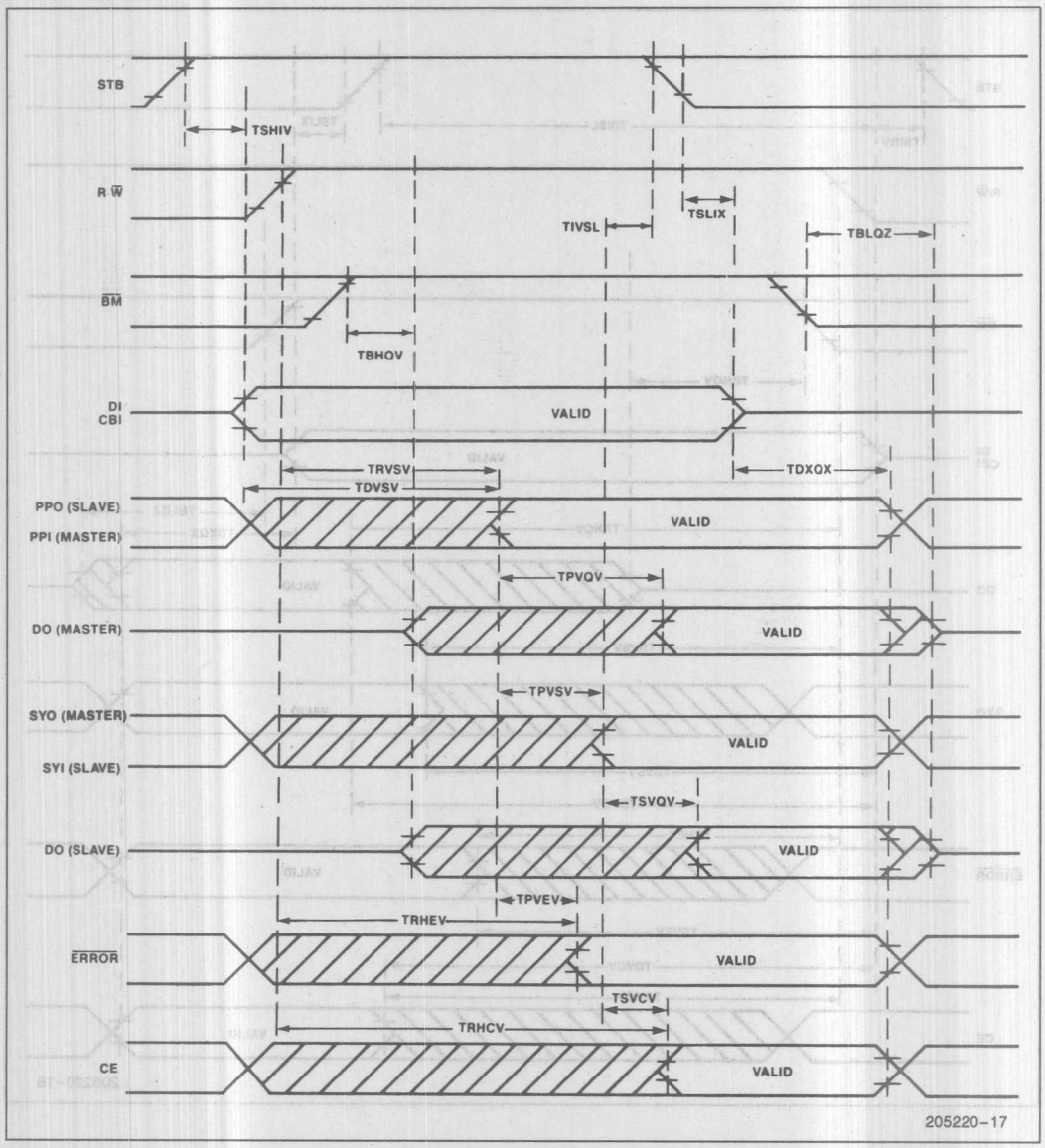
WAVEFORMS

READ



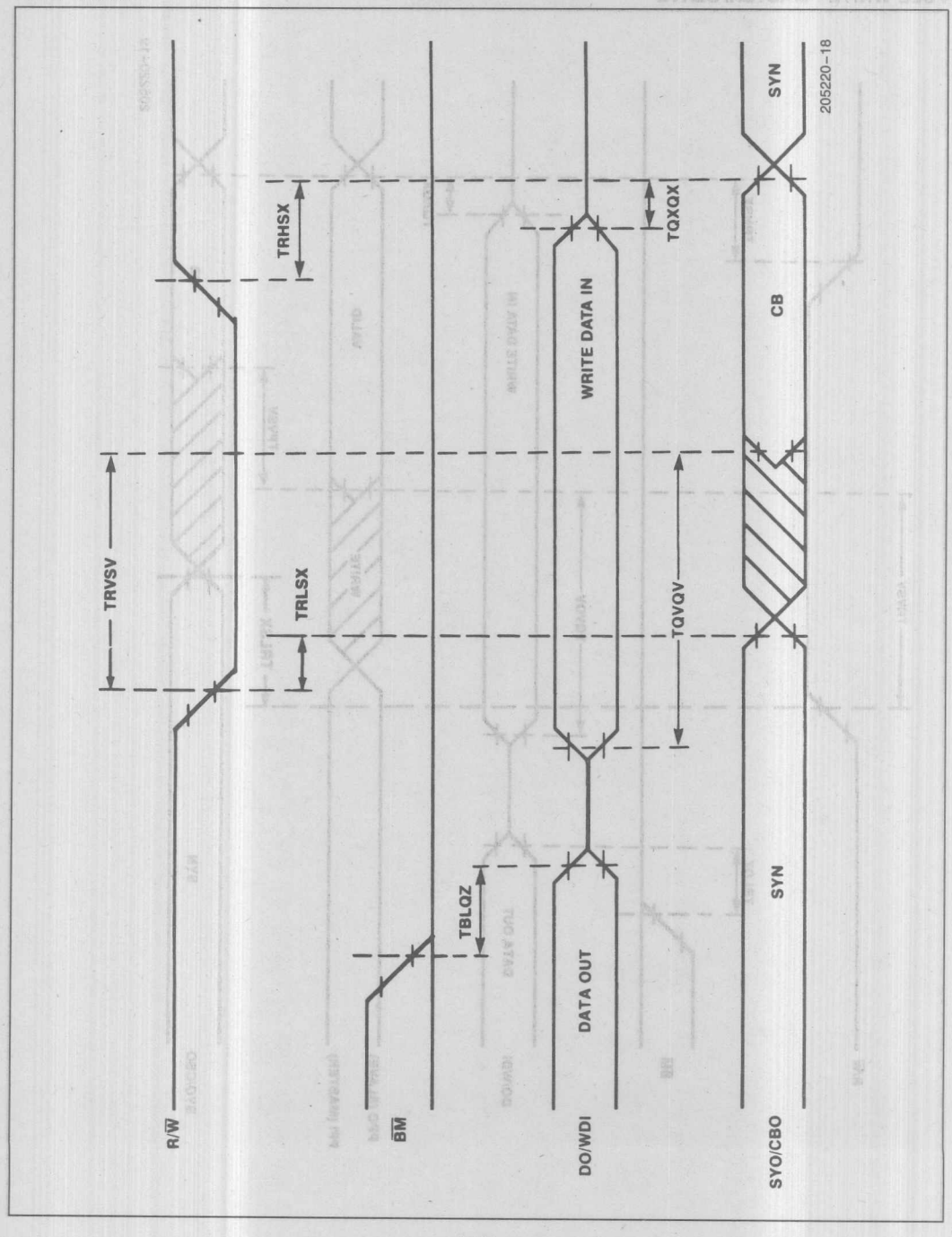
WAVEFORMS (Continued)

READ—MASTER/SLAVE



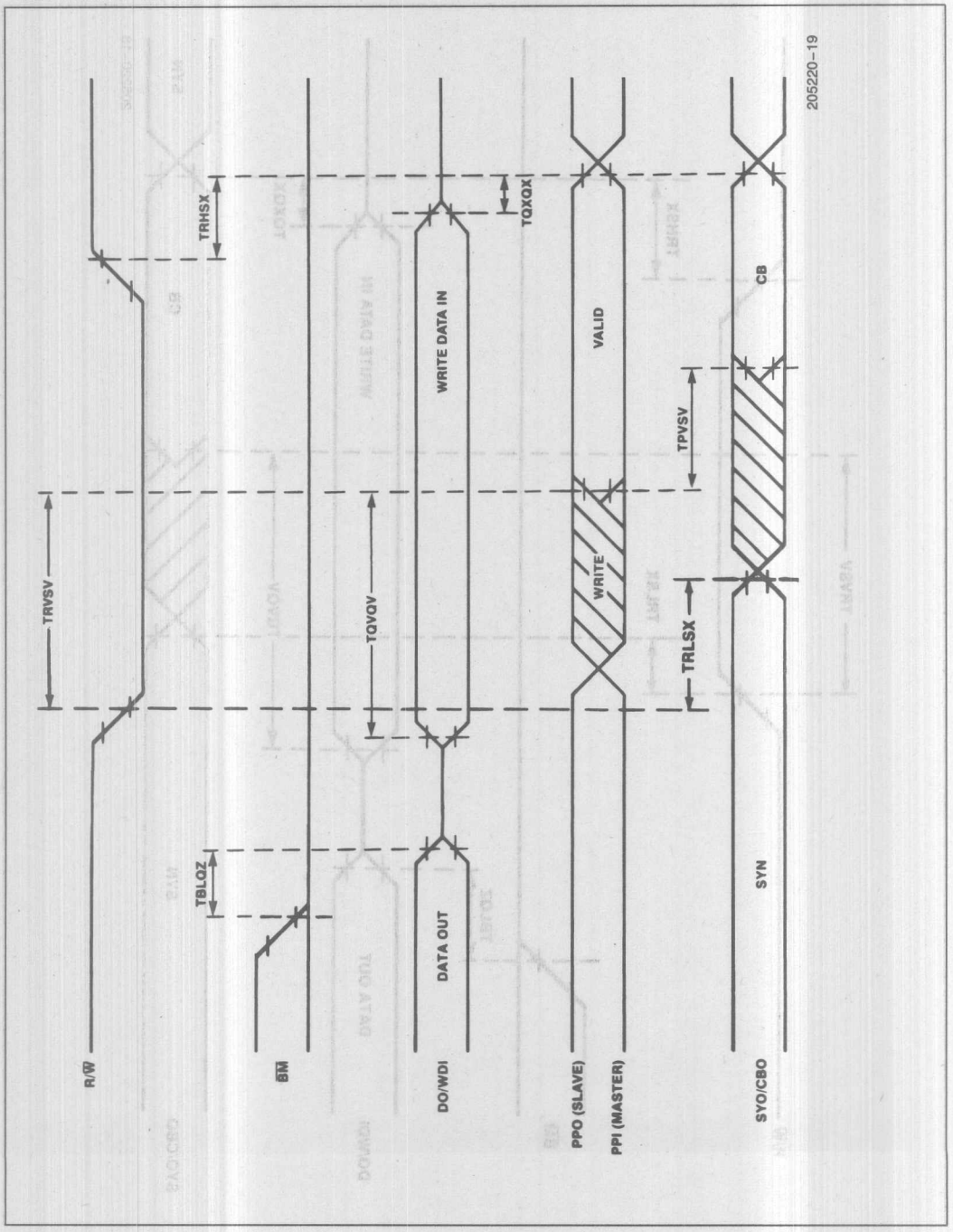
WAVEFORMS (Continued)

FULL WRITE



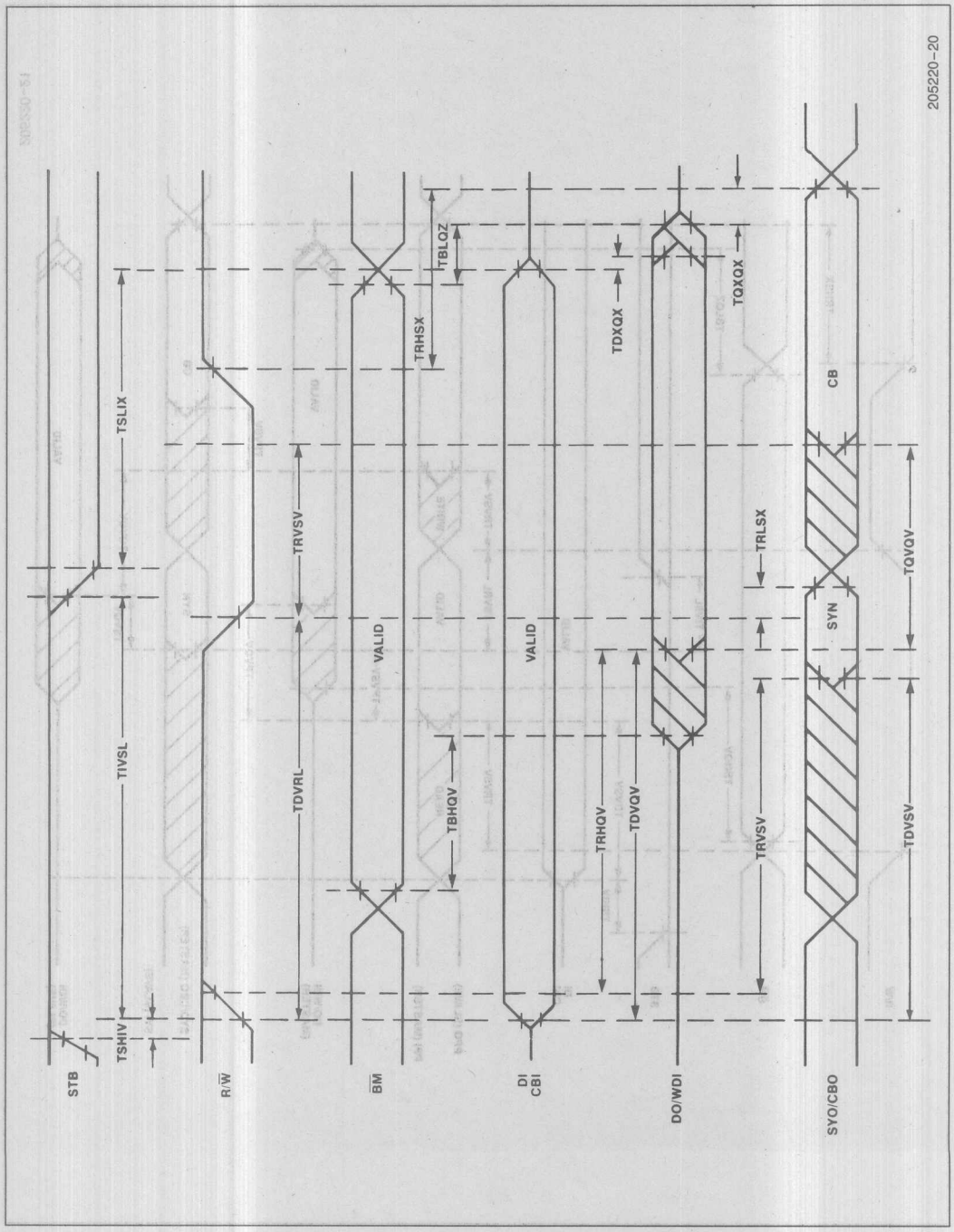
WAVEFORMS (Continued)

FULL WRITE—MASTER/SLAVE



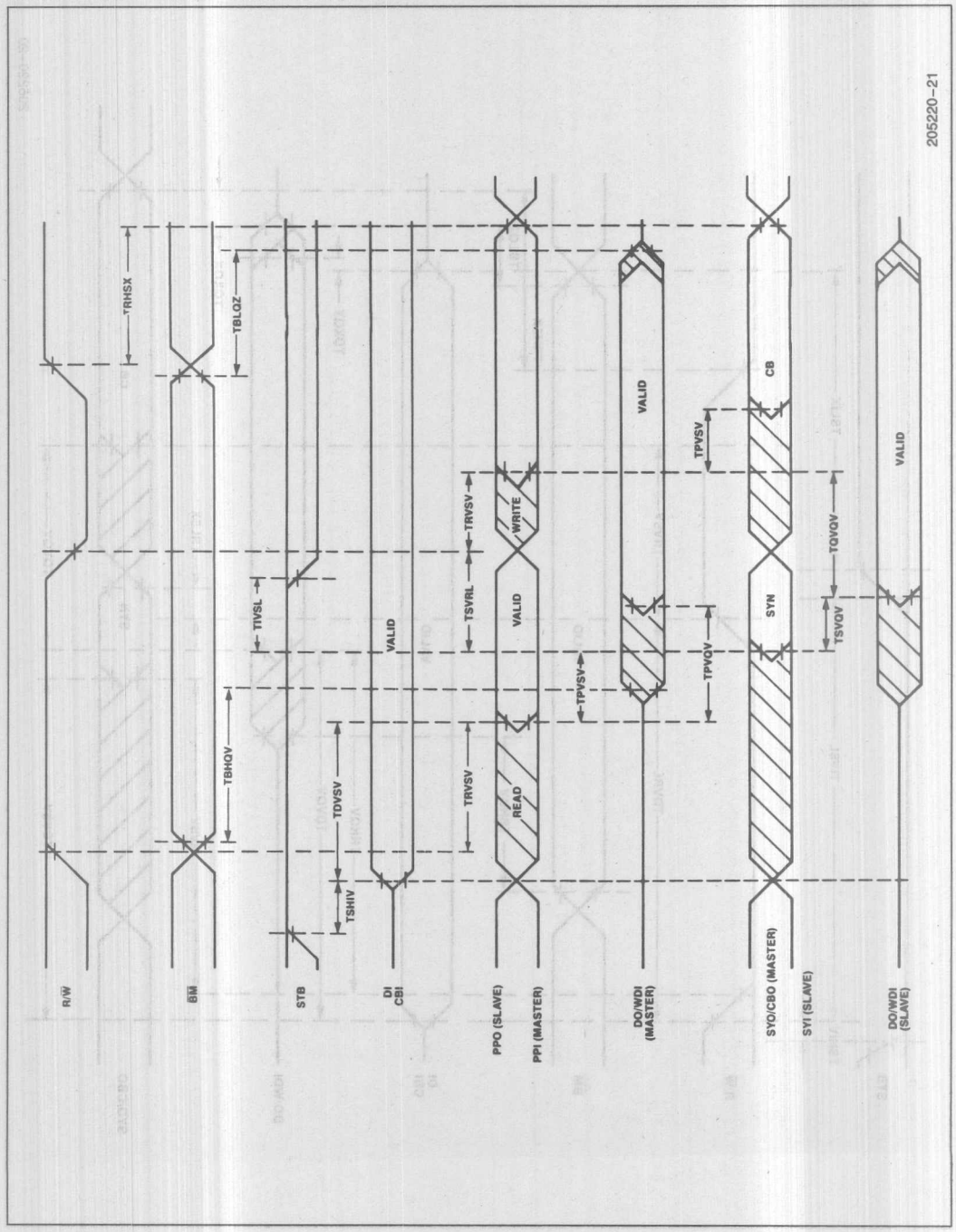
WAVEFORMS (Continued)

READ MODIFY WRITE



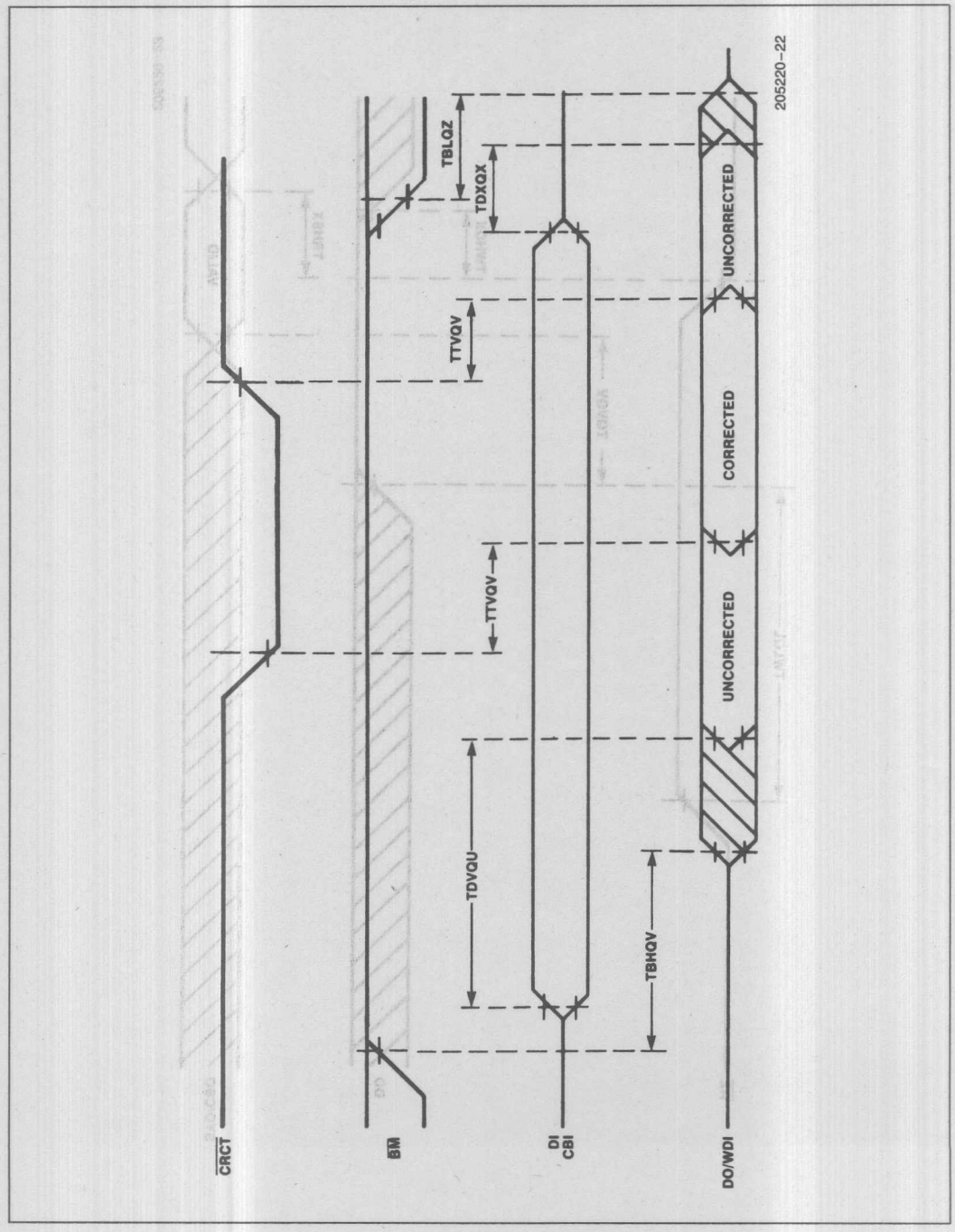
WAVEFORMS (Continued)

READ MODIFY WRITE—MASTER/SLAVE



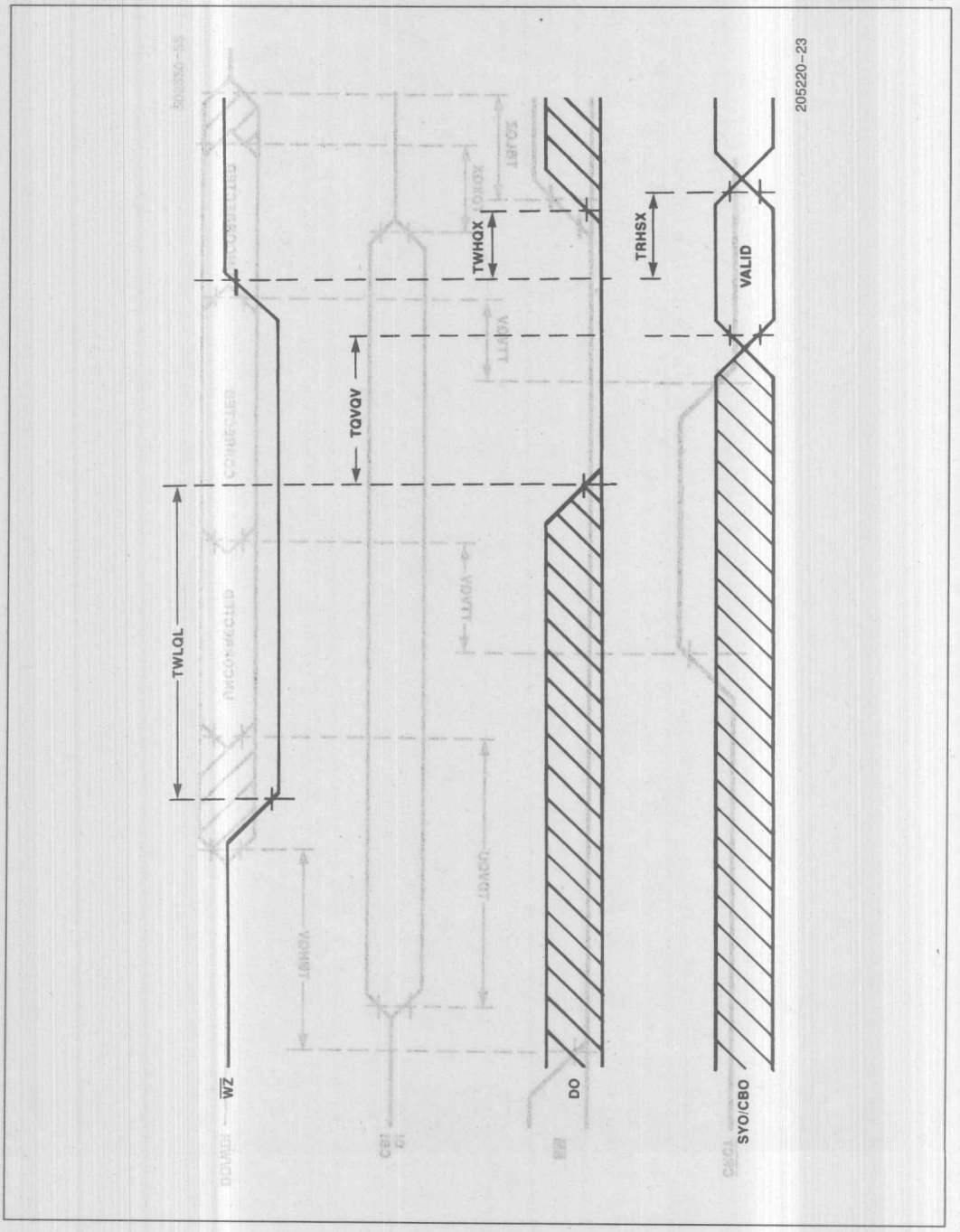
WAVEFORMS (Continued)

NON-CORRECTING READ



WAVEFORMS (Continued)

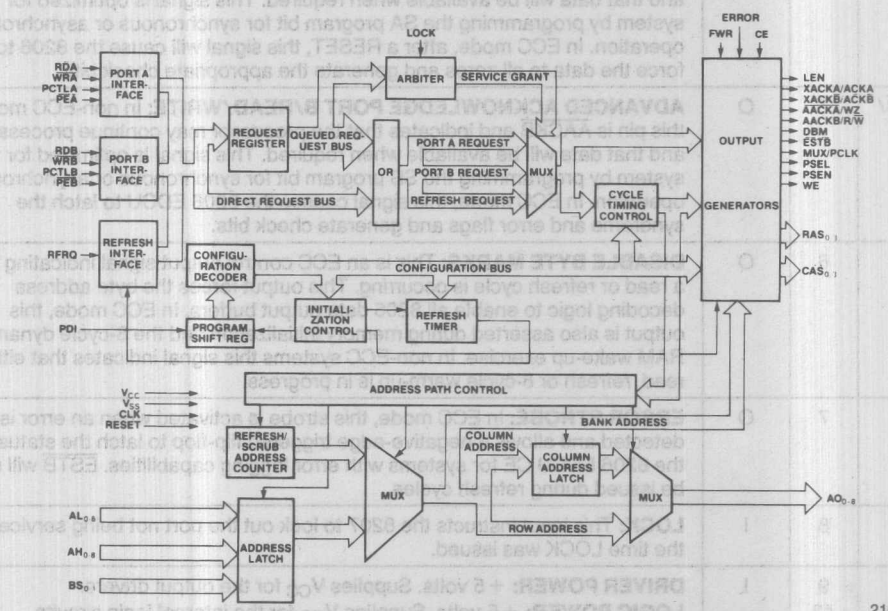
WRITE ZERO



8207
DUAL-PORT DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to Control 16K, 64K and 256K Dynamic RAMs**
- **Directly Addresses and Drives up to 2 Megabytes without External Drivers**
- **Supports Single and Dual-Port Configurations**
- **Automatic RAM Initialization in All Modes**
- **Four Programmable Refresh Modes**
- **Transparent Memory Scrubbing in ECC Mode**
- **Fast Cycle Support for 8 MHz 80286 with 8207-16**
- **Slow Cycle Support for 8 MHz, 10 MHz 8086/88, 80186/188 with 8207-8, 8207-10**
- **Provides Signals to Directly Control the 8206 Error Detection and Correction Unit**
- **Supports Synchronous or Asynchronous Operation on Either Port**
- **68 Lead JEDEC Type A Leadless Chip Carrier (LCC) and Pin Grid Array (PGA), Both in Ceramic.**

The Intel 8207 Dual-Port Dynamic RAM Controller is a high-performance, systems-oriented, Dynamic RAM controller that is designed to easily interface 16K, 64K and 256K Dynamic RAMs to Intel and other microprocessor systems. A dual-port interface allows two different busses to independently access memory. When configured with an 8206 Error Detection and Correction Unit the 8207 supplies the necessary logic for designing large error-corrected memory arrays. This combination provides automatic memory initialization and transparent memory error scrubbing.



210463-1

Figure 1. 8207 Block Diagram

September 1987
Order Number: 210463-007

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
LEN	1	O	**ADDRESS LATCH ENABLE:** In two-port configurations, when Port A is running with iAPX 286 Status interface mode, this output replaces the ALE signal from the system bus controller of port A and generates an address latch enable signal which provides optimum setup and hold timing for the 8207. This signal is used in Fast Cycle operation only.
$\overline{\text{XACKA}}$/ $\overline{\text{ACKA}}$	2	O	**TRANSFER ACKNOWLEDGE PORT A/ACKNOWLEDGE PORT A:** In non-ECC mode, this pin is $\overline{\text{XACKA}}$ and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port A. $\overline{\text{XACKA}}$ is a Multibus-compatible signal. In ECC mode, this pin is $\overline{\text{ACKA}}$ which can be configured, depending on the programming of the X program bit, as an $\overline{\text{XACK}}$ or $\overline{\text{AACK}}$ strobe. The SA programming bit determines whether the $\overline{\text{AACK}}$ will be an early $\overline{\text{EAACKA}}$ or a late $\overline{\text{LAACKA}}$ interface signal.
$\overline{\text{XACKB}}$/ $\overline{\text{ACKB}}$	3	O	**TRANSFER ACKNOWLEDGE PORT B/ACKNOWLEDGE PORT B:** In non-ECC mode, this pin is XACKB and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port B. $\overline{\text{XACKB}}$ is a Multibus-compatible signal. In ECC mode, this pin is $\overline{\text{ACKB}}$ which can be configured, depending on the programming of the X program bit, as an $\overline{\text{XACK}}$ or $\overline{\text{AACK}}$ strobe. The SB programming bit determines whether the $\overline{\text{AACK}}$ will be an early $\overline{\text{EAACKB}}$ or a late $\overline{\text{LAACKB}}$ interface signal.
$\overline{\text{AACKA}}$/ $\overline{\text{WZ}}$	4	O	**ADVANCED ACKNOWLEDGE PORT A/WRITE ZERO:** In non-ECC mode, this pin is $\overline{\text{AACKA}}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SA program bit for synchronous or asynchronous operation. In ECC mode, after a RESET, this signal will cause the 8206 to force the data to all zeros and generate the appropriate check bits.
$\overline{\text{AACKB}}$/ R/$\overline{\text{W}}$	5	O	**ADVANCED ACKNOWLEDGE PORT B/READ/WRITE:** In non-ECC mode, this pin is $\overline{\text{AACKB}}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SB program bit for synchronous or asynchronous operation. In ECC mode, this signal causes the 8206 EDCU to latch the syndrome and error flags and generate check bits.
$\overline{\text{DBM}}$	6	O	**DISABLE BYTE MARKS:** This is an ECC control output signal indicating that a read or refresh cycle is occurring. This output forces the byte address decoding logic to enable all 8206 data output buffers. In ECC mode, this output is also asserted during memory initialization and the 8-cycle dynamic RAM wake-up exercise. In non-ECC systems this signal indicates that either a read, refresh or 8-cycle warm-up is in progress.
$\overline{\text{ESTB}}$	7	O	**ERROR STROBE:** In ECC mode, this strobe is activated when an error is detected and allows a negative-edge triggered flip-flop to latch the status of the 8206 EDCU CE for systems with error logging capabilities. $\overline{\text{ESTB}}$ will not be issued during refresh cycles.
LOCK	8	I	**LOCK:** This input instructs the 8207 to lock out the port not being serviced at the time LOCK was issued.
V_{CC}	9 43	I	**DRIVER POWER:** +5 volts. Supplies V_{CC} for the output drivers. **LOGIC POWER:** +5 volts. Supplies V_{CC} for the internal logic circuits.
CE	10	I	**CORRECTABLE ERROR:** This is an ECC input from the 8206 EDCU which instructs the 8207 whether a detected error is correctable or not. A high input indicates a correctable error. A low input inhibits the 8207 from activating WE to write the data back into RAM. This should be connected to the CE output of the 8206.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
$\overline{\text{ERROR}}$	11	I	**ERROR:** This is an ECC input from the 8206 EDCU and instructs the 8207 that an error was detected. This pin should be connected to the ERROR output of the 8206.
MUX/ PCLK	12	O	**MULTIPLEXER CONTROL/PROGRAMMING CLOCK:** Immediately after a RESET this pin is used to clock serial programming data into the PDI pin. In normal two-port operation, this pin is used to select memory addresses from the appropriate port. When this signal is high, port A is selected and when it is low, port B is selected. This signal may change state before the completion of a RAM cycle, but the RAM address hold time is satisfied.
PSEL	13	O	**PORT SELECT:** This signal is used to select the appropriate port for data transfer. When this signal is high port A is selected and when it is low port B is selected.
PSEN	14	O	**PORT SELECT ENABLE:** This signal used in conjunction with PSEL provides contention-free port exchange on the data bus. When PSEN is low, port selection is allowed to change state.
WE	15	O	**WRITE ENABLE:** This signal provides the dynamic RAM array the write enable input for a write operation.
$\overline{\text{FWR}}$	16	I	**FULL WRITE:** This is an ECC input signal that instructs the 8207, in an ECC configuration, whether the present write cycle is normal RAM write (full write) or a RAM partial write (read-modify-write) cycle.
RESET	17	I	**RESET:** This signal causes all internal counters and state flip-flops to be reset and upon release of RESET, data appearing at the PDI pin is clocked in by the PCLK output. The states of the PDI, PCTLA, PCTLB and RFRQ pins are sampled by RESET going inactive and are used to program the 8207. An 8-cycle dynamic RAM warm-up is performed after clocking PDI bits into the 8207.
$\overline{\text{CAS0}}$–$\overline{\text{CAS3}}$	18–21	O	**COLUMN ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the column address, present on the AO0–8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{\text{RAS0}}$–$\overline{\text{RAS3}}$	22–25	O	**ROW ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the row address, present on the AO0–8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.
V_{SS}	26 60	I I	**DRIVER GROUND:** Provides a ground for the output drivers. **LOGIC GROUND:** Provides a ground for the remainder of the device.
AO0–AO8	35–27	O	**ADDRESS OUTPUTS:** These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.
BS0–BS1	36–37	I	**BANK SELECT:** These inputs are used to select one of four banks of the dynamic RAM array as defined by the program bits RB0 and RB1.
AL0–AL8	38–42 44–47	I	**ADDRESS LOW:** These lower-order address inputs are used to generate the row address for the internal address multiplexer.
AH0–AH8	48–56	I	**ADDRESS HIGH:** These higher-order address inputs are used to generate the column address for the internal address multiplexer.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
PDI	57	I	**PROGRAM DATA INPUT:** This input programs the various user-selectable options in the 8207. The PCLK pin shifts programming data into the PDI input from optional external shift registers. This pin may be strapped high or low to a default ECC (PDI = Logic "1") or non-ECC (PDI = Logic "0") mode configuration.
RFRQ	58	I	**REFRESH REQUEST:** This input is sampled on the falling edge of RESET. If it is high at RESET, then the 8207 is programmed for internal refresh request or external refresh request with failsafe protection. If it is low at RESET, then the 8207 is programmed for external refresh without failsafe protection or burst refresh. Once programmed the RFRQ pin accepts signals to start an external refresh with failsafe protection or external refresh without failsafe protection or a burst refresh.
CLK	59	I	**CLOCK:** This input provides the basic timing for sequencing the internal logic.
$\overline{\text{RDB}}$	61	I	**READ FOR PORT B:** This pin is the read memory request command input for port B. This input also directly accepts the $\overline{\text{S1}}$ status line from Intel processors.
$\overline{\text{WRB}}$	62	I	**WRITE FOR PORT B:** This pin is the write memory request command input for port B. This input also directly accepts the $\overline{\text{S0}}$ status line from Intel processors.
$\overline{\text{PEB}}$	63	I	**PORT ENABLE FOR PORT B:** This pin serves to enable a RAM cycle request for port B. It is generally decoded from the port address.
PCTLB	64	I	**PORT CONTROL FOR PORT B:** This pin is sampled on the falling edge of RESET. If low after RESET, the 8207 is programmed to accept memory read and write commands, Multibus commands or iAPX 286 status inputs. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The $\overline{\text{S2}}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be used as a Multibus-compatible inhibit signal.
$\overline{\text{RDA}}$	65	I	**READ FOR PORT A:** This pin is the read memory request command input for port A. This input also directly accepts the $\overline{\text{S1}}$ status line from Intel processors.
$\overline{\text{WRA}}$	66	I	**WRITE FOR PORT A:** This pin is the write memory request command input for port A. This input also directly accepts the $\overline{\text{S0}}$ status line from Intel processors.
$\overline{\text{PEA}}$	67	I	**PORT ENABLE FOR PORT A:** This pin serves to enable a RAM cycle request for port A. It is generally decoded from the port address.
PCTLA	68	I	**PORT CONTROL FOR PORT A:** This pin is sampled on the falling edge of RESET. If low after RESET, the 8207 is programmed to accept memory read and write commands, Multibus commands or iAPX 286 status inputs. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The $\overline{\text{S2}}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be connected to INHIBIT when operating with Multibus.

GENERAL DESCRIPTION

The Intel 8207 Dual-Port Dynamic RAM Controller is a microcomputer peripheral device which provides the necessary signals to address, refresh and directly drive 16K, 64K and 256K dynamic RAMs. This controller also provides the necessary arbitration circuitry to support dual-port access of the dynamic RAM array.

The 8207 supports several microprocessor interface options including synchronous and asynchronous connection to iAPX 86, iAPX 88, iAPX 186, iAPX 188, iAPX 286 and Multibus.

This device may be used with the 8206 Error Detection and Correction Unit (EDCU). When used with the 8206, the 8207 is programmed in the Error Checking and Correction (ECC) mode. In this mode, the 8207 provides all the necessary control signals for the 8206 to perform memory initialization and transparent error scrubbing during refresh.

FUNCTIONAL DESCRIPTION

Processor Interface

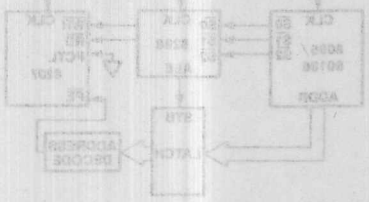
The 8207 has control circuitry for two ports each capable of supporting one of several possible bus structures. The ports are independently configurable allowing the dynamic RAM to serve as an interface between two different bus structures.

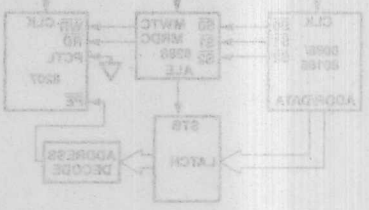
Each port of the 8207 may be programmed to run synchronous or asynchronous to the processor clock. (See Synchronous/Asynchronous Mode.) The 8207 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186, iAPX 188, and iAPX 286. When the 8207 is programmed to run in asynchronous mode, the 8207 inserts the necessary synchronization circuitry for the $\overline{RD}$, $\overline{WR}$, $\overline{PE}$, and PCTL inputs.

The 8207 achieves high performance (i.e., no wait states) by decoding the status lines directly from the iAPX 86, iAPX 88, iAPX 186, iAPX 188 and iAPX 286 processors. The 8207 can also be programmed to receive read or write Multibus commands or commands from a bus controller. (See Status/Command Mode.)

The 8207 may be programmed to accept the clock of the iAPX 86, 88, 186, 188 or 286. The 8207 adjusts its internal timing to allow for the different clock frequencies of these microprocessors. (See Microprocessor Clock Frequency Option.)

Figures 2A and 2B show the different processor interfaces to the 8207 using the synchronous or asynchronous mode and status or command interface.





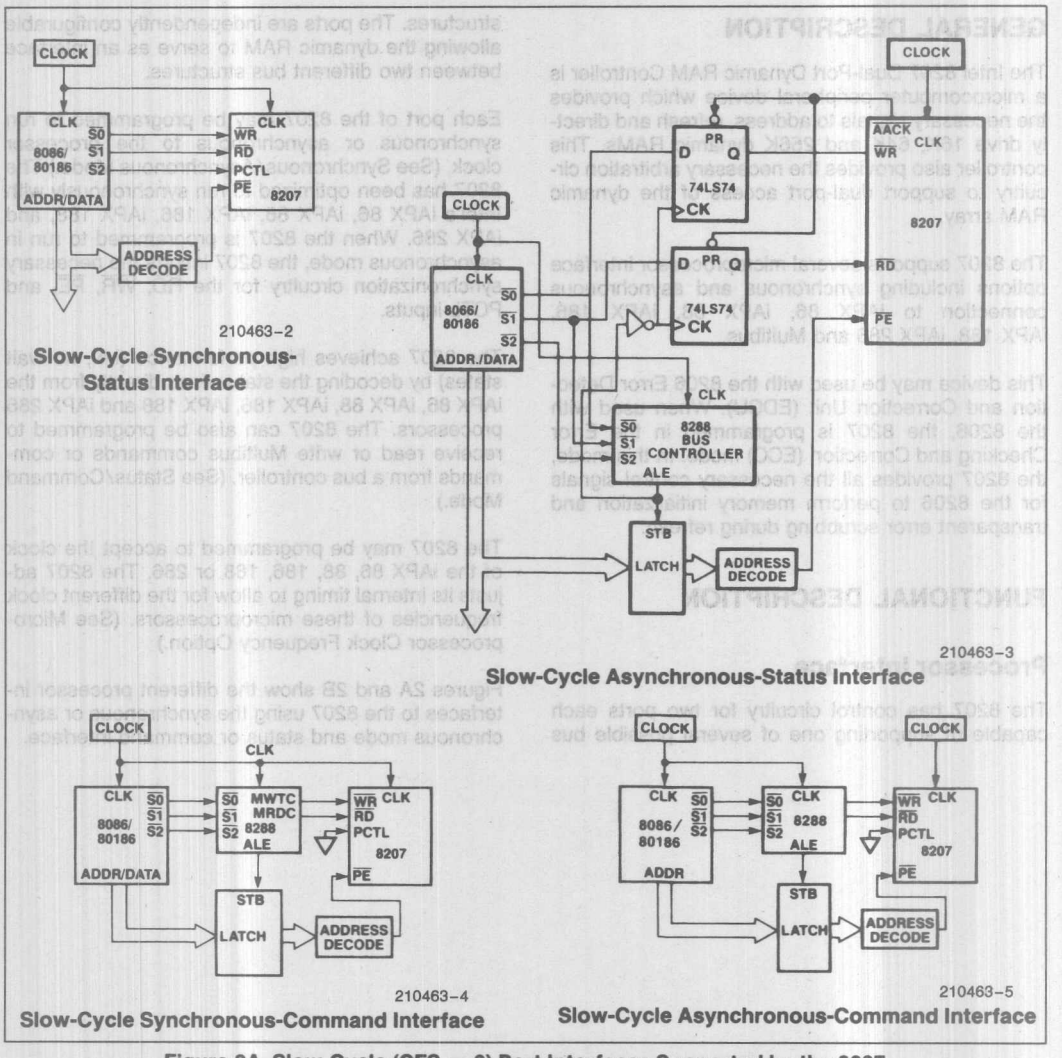


Figure 2A. Slow-Cycle (CFS = 0) Port Interfaces Supported by the 8207

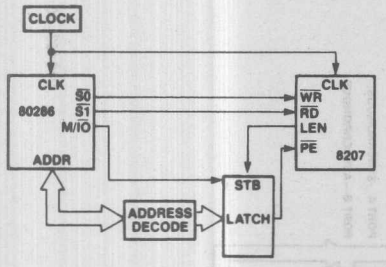
Single-Port Operation

The use of an address latch with the iAPX 286 status interface is not needed since the 8207 can internally latch the addresses with an internal signal similar in behavior to the LEN output. This operation is active only in single-port applications when the processor is interfaced to port A.

Dual-Port Operation

The 8207 provides for two-port operation. Two independent processors may access memory controlled by the 8207. The 8207 arbitrates between each of the processor requests and directs data to or from the appropriate port. Selection is done on a priority concept that reassigns priorities based upon past history. Processor requests are internally queued.

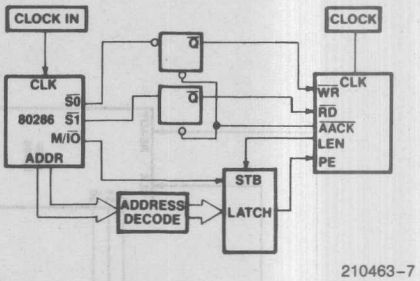
Figure 3 shows a dual-port configuration with two iAPX 86 systems interfacing to dynamic RAM. One of the processor systems is interfaced synchronously using the status interface and the other is interfaced asynchronously also using the status interface.



210463-6

NOTE:
Address latch not required in single-port mode.

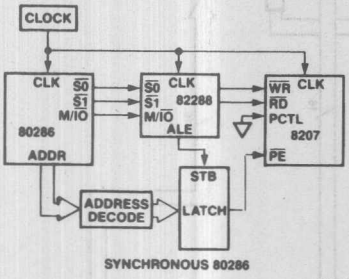
Fast-Cycle Synchronous-Status Interface



210463-7

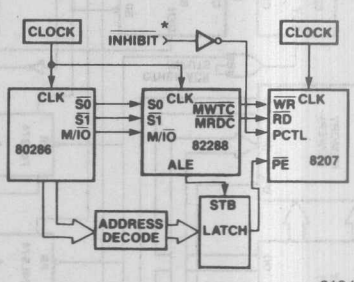
NOTE:
Address latch not required in single-port mode.

Fast-Cycle Asynchronous-Status Interface



210463-8

Fast-Cycle Synchronous-Command Interface



210463-9

*MULTI-BUS Option

Fast-Cycle Asynchronous-Command Interface

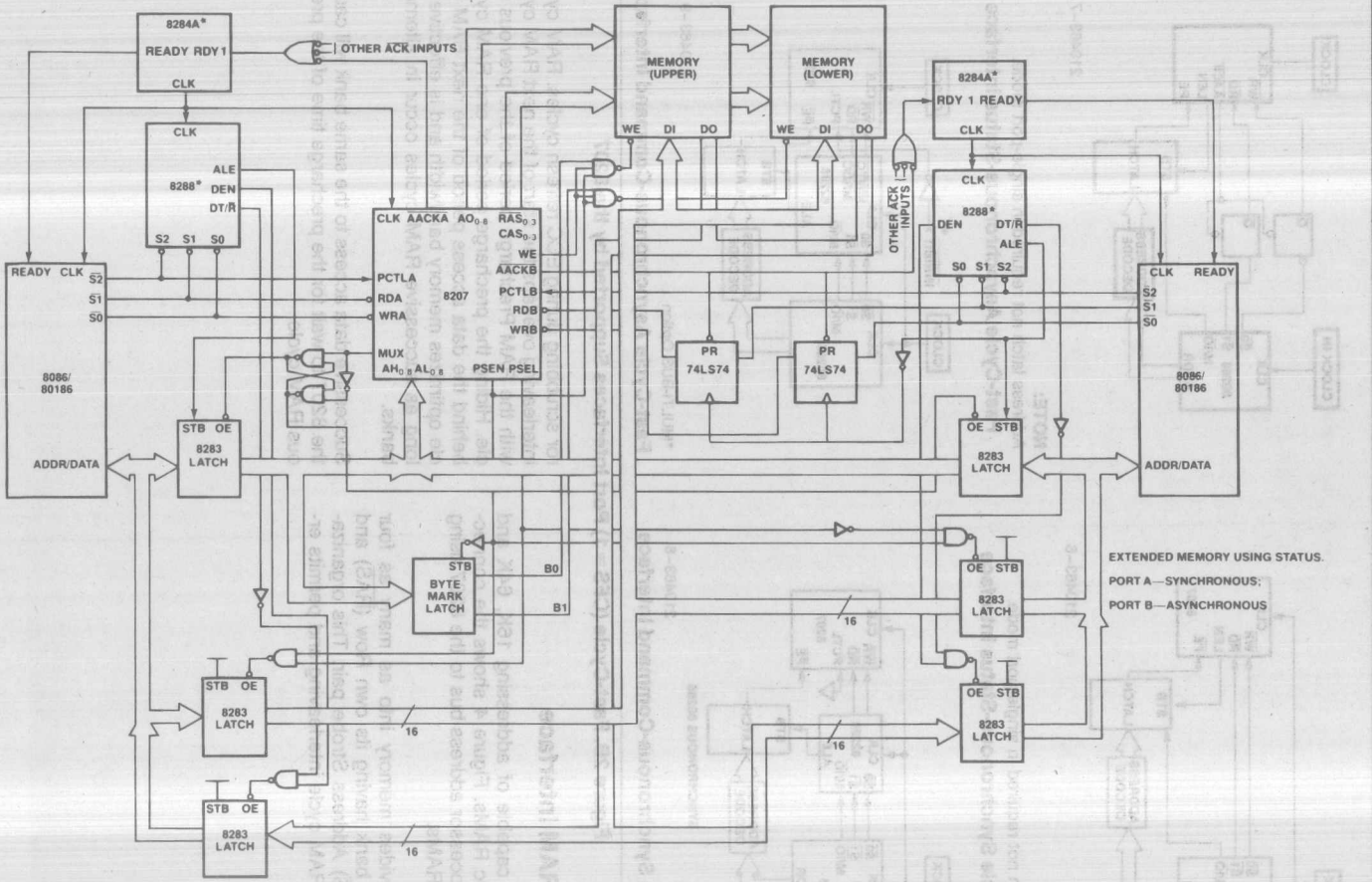
Figure 2B. Fast-Cycle (CFS = 1) Port Interfaces Supported by the 8207

Dynamic RAM Interface

The 8207 is capable of addressing 16K, 64K and 256K dynamic RAMs. Figure 4 shows the connection of the processor address bus to the 8207 using the different RAMs.

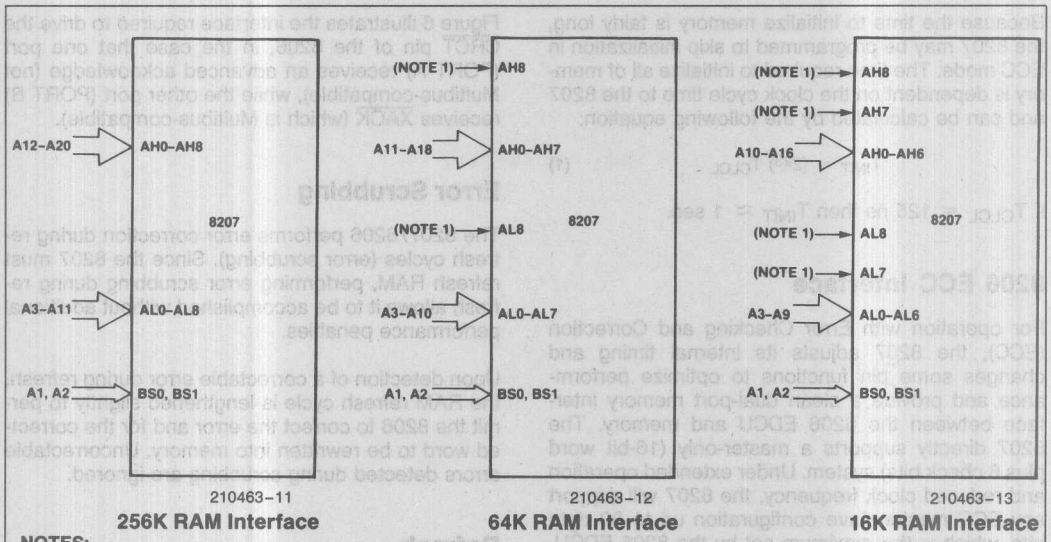
The 8207 divides memory into as many as four banks, each bank having its own Row ($\overline{RAS}$) and Column ($\overline{CAS}$) Address Strobe pair. This organization permits RAM cycle interleaving and permits error scrubbing during ECC refresh cycles. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM Precharge period of the previous cycle. Hiding the precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in alternate banks.

Successive data access to the same bank will cause the 8207 to wait for the precharge time of the previous RAM cycle.



NOTE:
*These components are not necessary when using the 80186 components. These functions are provided directly by the 80186.

Figure 3. 8086/80186 Dual Port System



NOTES:
1. Unassigned address input pins should be strapped high or low.
2. A0 along with BHE are used to select a byte within a processor word.
3. Low order address bits are used as bank select inupts so that consecutive memory access requests are to alternate banks allowing bank interleaving of memory cycles.

Figure 4. Processor Address Interface to the 8207 Using 16K, 64K, and 256K RAMs

If not all RAM banks are occupied, the 8207 reassigns the $\overline{RAS}$ and $\overline{CAS}$ strobes to allow using wider data words without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers. Table 2 shows the bank selection decoding and the word expansion, including $\overline{RAS}$ and $\overline{CAS}$ assignments. For example, if only two RAM banks are occupied, then two $\overline{RAS}$ and two $\overline{CAS}$ strobes are activated per bank. Program bits RB1 and RB0 are not used to check the bank select inputs BS1 and BS0. The system design must protect from accesses to "illegal", non-existent banks of memory, by deactivating the PEA, PEB inputs when addressing an illegal bank.

The 8207 can interface to fast or slow RAMs. The 8207 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option.)

Memory Initialization

After programming, the 8207 performs eight RAM "warm-up" cycles to prepare the dynamic RAM for proper device operation. During "warm-up" some RAM parameters, such as tRAH, tASC, may not be met. This causes no harm to the dynamic RAM array. If configured for operation with error correction, the 8207 and 8206 EDCU will proceed to initialize all of memory (memory is written with zeros with corresponding check bits).

Table 2. Bank Selection Decoding and Word Expansion

Program Bits		Bank Input		RAS/CAS Pair Allocation
RB1	RB0	BS1	BS0	
0	0	0	0	RAS_{0-3}, CAS_{0-3} to Bank 0
0	0	0	1	Illegal
0	0	1	0	Illegal
0	0	1	1	Illegal
0	1	0	0	$RAS_{0,1}$, $CAS_{0,1}$ to Bank 0
0	1	0	1	$RAS_{2,3}$, $CAS_{2,3}$ to Bank 1
0	1	1	0	Illegal
0	1	1	1	Illegal
1	0	0	0	RAS_0, CAS_0 to Bank 0
1	0	0	1	RAS_1, CAS_1 to Bank 1
1	0	1	0	RAS_2, CAS_2 to Bank 2
1	0	1	1	Illegal
1	1	0	0	RAS_0, CAS_0 to Bank 0
1	1	0	1	RAS_1, CAS_1 to Bank 1
1	1	1	0	RAS_2, CAS_2 to Bank 2
1	1	1	1	RAS_3, CAS_3 to Bank 3

Because the time to initialize memory is fairly long, the 8207 may be programmed to skip initialization in ECC mode. The time required to initialize all of memory is dependent on the clock cycle time to the 8207 and can be calculated by the following equation:

$$T_{INIT} = (2^{23})\, T_{CLCL} \quad (1)$$

if $T_{CLCL} = 125$ ns then $T_{INIT} \approx 1$ sec.

8206 ECC Interface

For operation with Error Checking and Correction (ECC), the 8207 adjusts its internal timing and changes some pin functions to optimize performance and provide a clean dual-port memory interface between the 8206 EDCU and memory. The 8207 directly supports a master-only (16-bit word plus 6 check bits) system. Under extended operation and reduced clock frequency, the 8207 will support any ECC master-slave configuration up to 80 data bits, which is the maximum set by the 8206 EDCU. (See Extend Option.)

Correctable errors detected during memory read cycles are corrected immediately and then written back into memory.

In a synchronous bus environment, ECC system performance has been optimized to enhance processor throughput, while in an asynchronous bus environment (the Multibus), ECC performance has been optimized to get valid data onto the bus as quickly as possible. Performance optimization, processor throughput or quick data access may be selected via the Transfer Acknowledge Option.

The main difference between the two ECC implementations is that, when optimized for processor throughput, RAM data is always corrected and an advanced transfer acknowledge is issued at a point when, by knowing the processor characteristics, data is guaranteed to be valid by the time the processor needs it.

When optimized for quick data access, (valid for Multibus) the 8206 is configured in the uncorrecting mode where the delay associated with error correction circuitry is transparent, and a transfer acknowledge is issued as soon as valid data is known to exist. If the $\overline{\text{ERROR}}$ flag is activated, then the transfer acknowledge is delayed until after the 8207 has instructed the 8206 to correct the data and the corrected data becomes available on the bus. Figure 5 illustrates a dual-port ECC system.

Figure 6 illustrates the interface required to drive the $\overline{\text{CRCT}}$ pin of the 8206, in the case that one port (PORT A) receives an advanced acknowledge (not Multibus-compatible), while the other port (PORT B) receives $\overline{\text{XACK}}$ (which is Multibus-compatible).

Error Scrubbing

The 8207/8206 performs error correction during refresh cycles (error scrubbing). Since the 8207 must refresh RAM, performing error scrubbing during refresh allows it to be accomplished without additional performance penalties.

Upon detection of a correctable error during refresh, the RAM refresh cycle is lengthened slightly to permit the 8206 to correct the error and for the corrected word to be rewritten into memory. Uncorrectable errors detected during scrubbing are ignored.

Refresh

The 8207 provides an internal refresh interval counter and a refresh address counter to allow the 8207 to refresh memory. The 8207 will refresh 128 rows every 2 milliseconds or 256 rows every 4 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 8207 may be programmed for any of four different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst Refresh mode, or no refresh. (See Refresh Options.)

It is possible to decrease the refresh time interval by 10%, 20% or 30%. This option allows the 8207 to compensate for reduced clock frequencies. Note that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options.)

External Refresh Requests after RESET

External refresh requests are not recognized by the 8207 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper

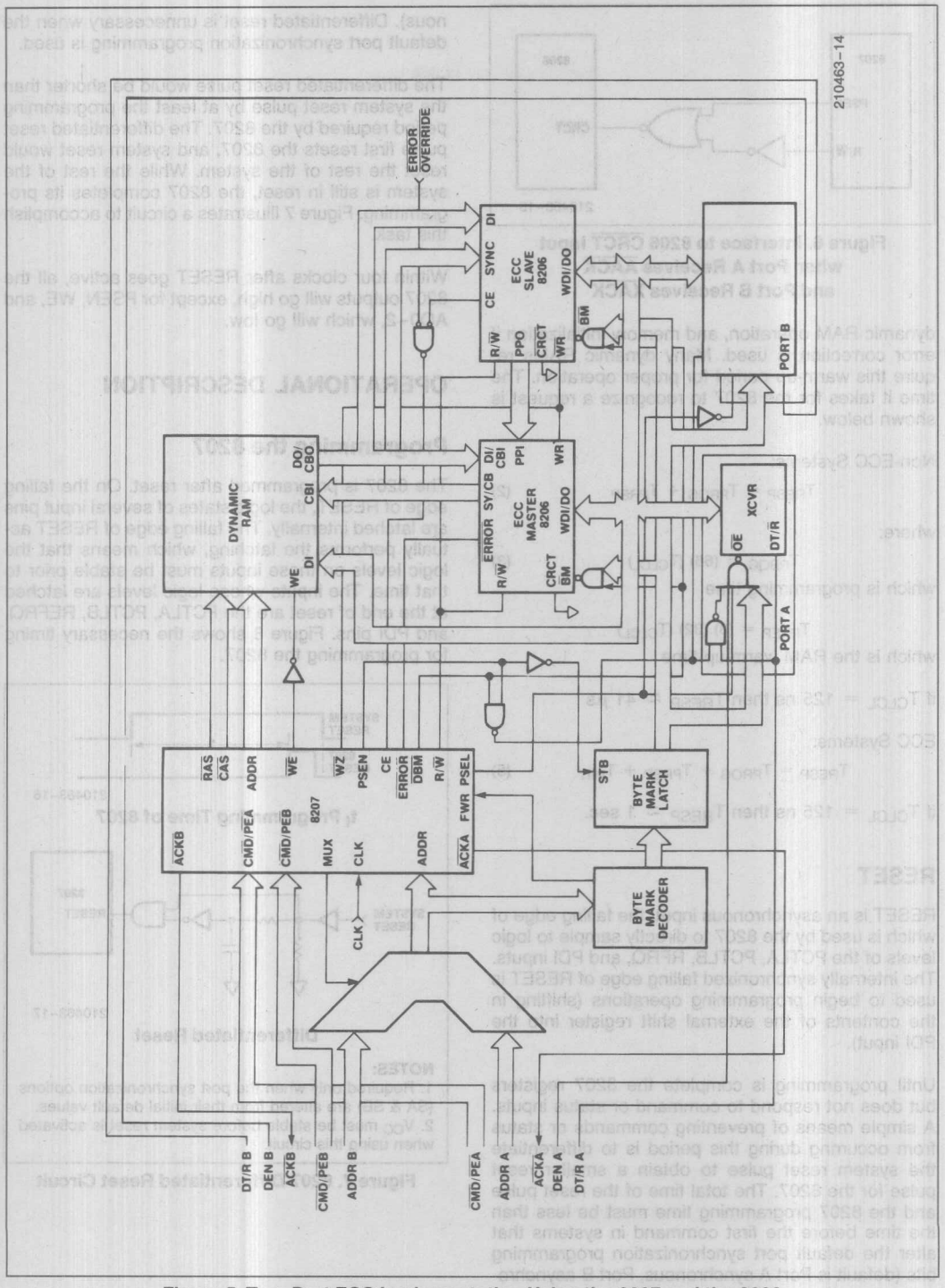


Figure 5. Two-Port ECC Implementation Using the 8207 and the 8206

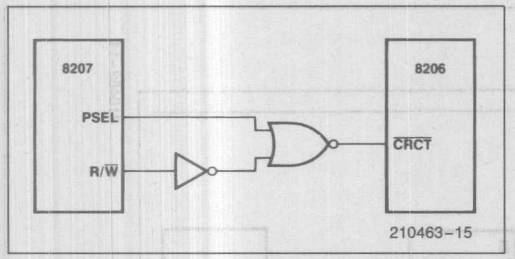


Figure 6. Interface to 8206 $\overline{\text{CRCT}}$ Input when Port A Receives $\overline{\text{AACK}}$ and Port B Receives $\overline{\text{XACK}}$

dynamic RAM operation, and memory initialization if error correction is used. Many dynamic RAMs require this warm-up period for proper operation. The time it takes for the 8207 to recognize a request is shown below.

Non-ECC Systems:

$$T_{RESP} = T_{PROG} + T_{PREP} \quad (2)$$

where:

$$T_{PROG} = (66)\ (T_{CLCL}) \quad (3)$$

which is programming time

$$T_{PREP} = (8)\ (32)\ (T_{CLCL})$$

which is the RAM warm-up time

if $T_{CLCL} = 125$ ns then $T_{RESP} \approx 41\ \mu s$

ECC Systems:

$$T_{RESP} = T_{PROG} + T_{PREP} + T_{INIT} \quad (5)$$

if $T_{CLCL} = 125$ ns then $T_{RESP} \approx 1$ sec.

RESET

RESET is an asynchronous input, the falling edge of which is used by the 8207 to directly sample to logic levels of the PCTLA, PCTLB, RFRQ, and PDI inputs. The internally synchronized falling edge of RESET is used to begin programming operations (shifting in the contents of the external shift register into the PDI input).

Until programming is complete the 8207 registers but does not respond to command or status inputs. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 8207. The total time of the reset pulse and the 8207 programming time must be less than the time before the first command in systems that alter the default port synchronization programming bits (default is Port A synchronous, Port B asynchronous). Differentiated reset is unnecessary when the default port synchronization programming is used.

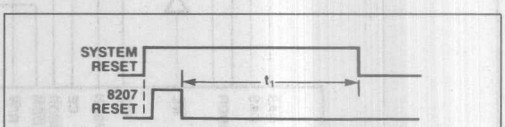
The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 8207. The differentiated reset pulse first resets the 8207, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 8207 completes its programming. Figure 7 illustrates a circuit to accomplish this task.

Within four clocks after RESET goes active, all the 8207 outputs will go high, except for PSEN, WE, and AO0–2, which will go low.

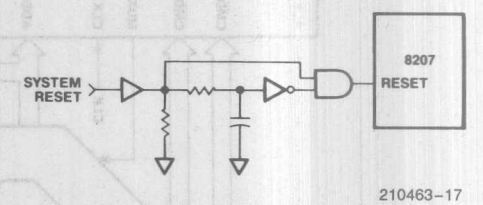
OPERATIONAL DESCRIPTION

Programming the 8207

The 8207 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTLA, PCTLB, REFRQ, and PDI pins. Figure 8 shows the necessary timing for programming the 8207.



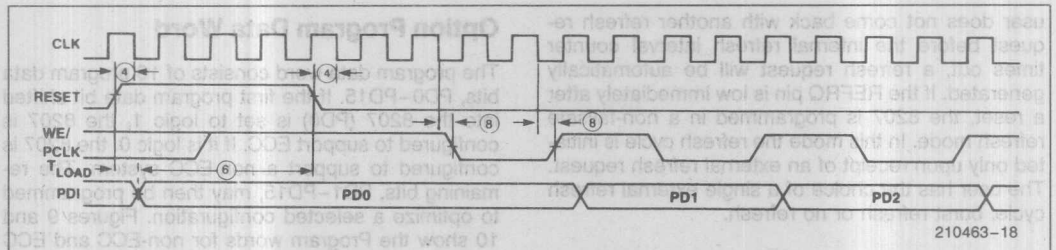
t_I Programming Time of 8207



Differentiated Reset

NOTES:
1. Required only when the port synchronization options (SA & SB) are altered from their initial default values.
2. V_{CC} must be stable before system reset is activated when using this circuit.

Figure 7. 8207 Differentiated Reset Circuit



NOTES:
TRTVCL—Reset is an asynchronous input, if reset occurs before T_1, then it is guaranteed to be recognized.
TPGVCL—Minimum PDI valid time prior to reset going low.
TCLPC—MUX/PCLK delay.
TLOAD—Asynchronous load data propagation delay.

Figure 8. Timing Illustrating External Shift Register Requirements for Programming the 8207

Status/Command Mode

The two processor ports of the 8207 are configured by the states of the PCTLA and PCTLB pins. Which interface is selected depends on the state of the individual port's PCTL pin at the end of reset. If PCTL is high at the end of the reset, the 8086 Status interface is selected; if it is low, then the Command interface is selected.

The status lines of the 80286 are similar in code and timing to the Multibus command lines, while the status code and timing of the 8086 and 8088 are identical to those of the 80186 and 80188 (ignoring the differences in clock duty cycle). Thus there exists two interface configurations, one for the 80286 status or Multibus memory commands, which is called the Command interface, and one for 8086, 8088, 80186 or 80188 status, called the 8086 Status interface. The Command interface can also directly interface to the command lines of the bus controllers for the 8086, 8088, 80186 and the 80286.

The 8086 Status interface allows direct decoding of the status of the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded. While in the Command mode the iAPX 286 status can be directly decoded. Microprocessor bus controller read or write commands or Multibus commands can also be directed to the 8207 when in Command mode.

Refresh Options

Immediately after system reset, the state of the REFRQ input pin is examined. If REFRQ is high, the 8207 provides the user with the choice between self-refresh or user-generated refresh with failsafe protection. Failsafe protection guarantees that if the

Table 3A. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286
0	0	0	Interrupt	Interrupt
0	0	1	I/O Read	I/O Read
0	1	0	I/O Write	I/O Write
0	1	1	Halt	Idle
1	0	0	Instruction Fetch	Halt
1	0	1	Memory Read	Memory Read
1	1	0	Memory Write	Memory Write
1	1	1	Idle	Idle

Table 3B. 8207 Response

8207 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086/80186 Status Interface	80286 Status or Command Interface
0	0	0	Ignore	Ignore*
0	0	1	Ignore	Read
0	1	0	Ignore	Write
0	1	1	Ignore	Ignore*
1	0	0	Read	Ignore
1	0	1	Read	Inhibit
1	1	0	Write	Inhibit
1	1	1	Ignore	Ignore

*Illegal with CFS = 0

user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the REFRQ pin is low immediately after a reset, the 8207 is programmed in a non-failsafe refresh mode. In this mode the refresh cycle is initiated only upon receipt of an external refresh request. The user has the choice of a single external refresh cycle, burst refresh or no refresh.

Internal Refresh Only

For the 8207 to generate internal refresh requests, it is necessary only to strap the REFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the REFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 8207. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold REFRQ low until after reset. Thereafter, bringing REFRQ high for one clock period causes refresh request to be generated. A refresh request is not recognized unitl a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., REFRQ is kept low until after reset). Thereafter, bringing REFRQ high for at least two clock periods causes a burst of up to 128 row address locations to be refreshed.

The ECC-configured systems, 128 locations are scrubbed. Any refresh request is not recognized until a previous request has been serviced (i.e., burst completed).

No Refresh

It is necessary to hold REFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping REFRQ low.

Option Program Data Word

The program data word consists of 16 program data bits, PD0–PD15. If the first program data bit shifted into the 8207 (PD0) is set to logic 1, the 8207 is configured to support ECC. If it is logic 0, the 8207 is configured to support a non-ECC system. The remaining bits, PD1–PD15, may then be programmed to optimize a selected configuration. Figures 9 and 10 show the Program words for non-ECC and ECC operation.

Using an External Shift Register

The 8207 may be configured to use an external shift register with asynchronous load capability such as a 74LS165. The reset pulse serves to parallel load the shift register and the 8207 supplies the clocking signal to shift the data in. Figure 11 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 8207 via the PDI pin (57), and clock is provided by the MUX/PCLK pin (12), which generates a total of 16 clock pulses. After programming is complete, data appearing at the input of the PDI pin is ignored. MUX/PCLK is a dual-function pin. During programming, it serves to clock the external shift register, and after programming is completed, it reverts to a MUX conrol pin. As the pin changes state to select different port addresses, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 8 illustrates the timing requirements of the shift register circuitry.

ECC Mode (ECC Program Bit)

The state of PDI (Program Data In) pin at reset determines whether the system is an ECC or non-ECC configuration. It is used internally by the 8207 to begin configuring timing circuits, even before programming is completely finished. The 8207 then begins programming the rest of the options.

Default Programming Options

After reset, the 8207 serially shifts in a program data word via the PDI pin. This pin may be strapped either high or low, or connected to an external shift register. Strapping PDI high causes the 8207 to default to a particular system configuration with error correction, and strapping it low causes the 8207 to default to a particular system configuration without error correction. Table 4 shows the default configurations.

PD15								PD8 PD7							PD0
0	0	TM 1	$\overline{PPR}$	$\overline{FFS}$	EXT	$\overline{PLS}$	CI0	CI1	$\overline{RB1}$	$\overline{RB0}$	$\overline{RFS}$	$\overline{CFS}$	SB	$\overline{SA}$	0

Program Data Bit	Name	Polarity/Function
PD0	ECC	ECC = 0 For Non-ECC Mode
PD1	$\overline{SA}$	$\overline{SA}$ = 0 Port A is Synchronous $\overline{SA}$ = 1 Port A is Asnychronous
PD2	SB	SB = 0 Port B is Asynchronous SB = 1 Port B is Synchronous
PD3	$\overline{CFS}$	$\overline{CFS}$ = 0 Fast-Cycle iAPX 286 Mode $\overline{CFS}$ = 1 Slow-Cycle iAPX 86 Mode
PD4	$\overline{RFS}$	$\overline{RFS}$ = 0 Fast RAM $\overline{RFS}$ = 1 Slow RAM
PD5 PD6	$\overline{RB0}$ $\overline{RB1}$	RAM Bank Occupancy See Table 2
PD7 PD8	CI1 CI0	Count Interval Bit 1; see Table 6 Count Interval Bit 0; see Table 6
PD9	$\overline{PLS}$	$\overline{PLS}$ = 0 Long Refresh Period $\overline{PLS}$ = 1 Short Refresh Period
PD10	EXT	EXT = 0 Not Extended EXT = 1 Extended
PD11	$\overline{FFS}$	$\overline{FFS}$ = 0 Fast CPU Frequency $\overline{FFS}$ = 1 Slow CPU Frequency
PD12	$\overline{PPR}$	$\overline{PPR}$ = 0 Most Recently Used Port Priority $\overline{PPR}$ = 1 Port A Preferred Priority
PD13	TM1	TM1 = 0 Test Mode 1 Off TM1 = 1 Test Mode 1 Enabled
PD14	0	Reserved, Must be Zero
PD15	0	Reserved, Must be Zero

Figure 9. Non-ECC Mode Program Data Word

PD15							PD8 PD7							PD0
$\overline{TM2}$	RB1	RB0	PPR	FFS	$\overline{EXT}$	PLS	$\overline{CI0}$	$\overline{CI1}$	XB	$\overline{XA}$	RFS	CFS	$\overline{SB}$	SA	1

Program Data Bit	Name	Polarity/Function
PD0	ECC	ECC = 1 ECC Mode
PD1	SA	SA = 0 Port A Asynchronous SA = 1 Port A Synchronous
PD2	$\overline{SB}$	$\overline{SB}$ = 0 Port B Synchronous $\overline{SB}$ = 1 Port B Asynchronous
PD3	CFS	CFS = 0 Slow-Cycle iAPX 86 Mode CFS = 1 Fast-Cycle iAPX 286 Mode
PD4	RFS	RFS = 0 Slow RAM RFS = 1 Fast RAM
PD5	$\overline{XA}$	$\overline{XA}$ = 0 MULTIBUS-Compatible ACKA $\overline{XA}$ = 1 Advanced ACKA Not Multibus-Compatible
PD6	XB	Advanced ACKB Not Multibus-Compatible XB = 1 Multibus-Compatible ACKB
PD7 PD8	$\overline{CI1}$ $\overline{CI0}$	Count Interval Bit 1; see Table 6 Count Interval Bit 0; see Table 6
PD9	PLS	PLS = 0 Short Refresh Period PLS = 1 Long Refresh Period
PD10	$\overline{EXT}$	$\overline{EXT}$ = 0 Master and Slave EDCU $\overline{EXT}$ = 1 Master EDCU Only
PD11	FFS	FFS = 0 Slow CPU Frequency FFS = 1 Fast CPU Frequency
PD12	PPR	PPR = 0 Port A Preferred Priority PPR = 1 Most Recently Used Port Priority
PD13 PD14	RB0 RB1	RAM Bank Occupancy See Table 2
PD15	$\overline{TM2}$	$\overline{TM2}$ = 0 Test Mode 2 Enabled $\overline{TM1}$ = 1 Test Mode 2 Off

Figure 10. ECC Mode Program Data Word

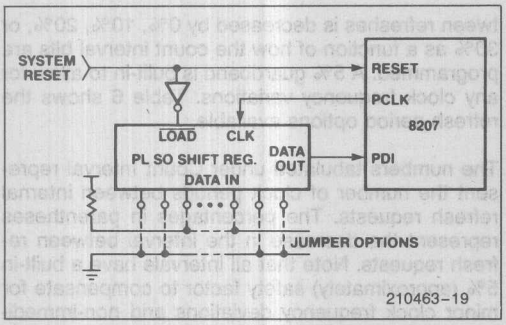


Figure 11. External Shift Register Interface

Table 4A. Default Non-ECC Programming, PDI Pin (57) Tied to Ground

Port A is Synchronous ($\overline{EAACKA}$ and $\overline{XACKA}$)
Port B is Asynchronous ($\overline{LAACKB}$ and $\overline{XACKB}$)
Fast-Cycle Processor Interface (iAPX 286)
Fast RAM
Refresh Interval uses 236 Clocks
128 Row Refresh in 2 ms; 256 Row Refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

Table 4B. Default ECC Programming, PDI Pin (57) Tied to V_{CC}

Port A is Synchronous
Port B is Asynchronous
Fast-Cycle Processor Interface (iAPX 286)
Fast RAM
Port A has $\overline{EAACKA}$ strobe (non-multibus)
Port B has $\overline{XACKB}$ strobe (multibus)
Refresh interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Master EDCU only (16-bit system)
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks ocuppied

If further system flexibility is needed, one or two external shift registers can be used to tailor the 8207 to its operating environment.

Synchronous/Asynchronous Mode (SA and SB Program Bits)

Each port of the 8207 may be independently configured to accept synchronous or asynchronous port commands ($\overline{RD}$, $\overline{WR}$, PCTL) and Port Enable ($\overline{PE}$) via the program bits SA and SB. The state of the SA and SB programming bits determine whether their associated ports are synchronous or asynchronous.

While a port may be configured with either the Status or Command interface in the synchronous mode, certain restrictions exist in the asynchronous mode. An asynchronous Command interface using the control lines of the Multibus is supported, and an asynchronous 8086 interface using the control lines of the 8086 is supported, with the use of TTL gates as illustrated in Figure 2. In the 8086 case, the TTL gates are needed to guarantee that status does not appear at the 8207's inputs too much before address, so that a cycle would start before address was valid.

Microprocessor Clock Frequency Option (CFS and FFS Program Bits)

The 8207 can be programmed to interface with slow-cycle microprocessors like the 8086, 8088, 81088 and 80186 or fast-cycle microprocessors like the 80286. The CFS bit configures the microprocessor interface to accept slow or fast cycle signals from either microprocessor group.

The FFS bit is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed.

Table 5. Microprocessor Clock Frequency Options

Program Bits		Processor	Clock Frequency
CFS	FFS		
0	0	iAPX 86, 88, 186, 188	≤ 6 MHz
0	1	iAPX 86, 88, 186, 188	> 6 MHz
1	0	iAPX 286	≤12 MHz
1	1	iAPX 286	>12 MHz

The external clock frequency must be programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

RAM Speed Option (RFS Program Bit)

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM.

Refresh Period Options (CI0, CI1 and PLS Program Bits)

The 8207 refreshes with either 128 rows every 2 milliseconds or 256 rows every 4 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option. The 7.8 microsecond refresh request rate is intended for those RAMs, 64K and above, which may require a faster refresh rate.

In addition to PLS program option, two other programming bits for refresh exist: Count Interval 0 (CI0) and Count Interval 1 (CI1). These two programming bits allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the same 15.6 or 7.8 microsecond period when the 8207 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests. Note that all intervals have a built-in 5% (approximately) safety factor to compensate for minor clock frequency deviations and non-immediate response to internal refresh requests.

Extend Option (EXT Program Bit)

The Extend option lengthens the memory cycle to allow longer access time which may be required by the system. Extend alters the RAM timing to compensate for increased loading on the Row and Column Address Strobes, and in the multiplexed Address Out lines.

Port Priority Option and Arbitration (PPR Program Bit)

The 8207 has to internally arbitrate among three ports: Port A, Port B and Port C—the refresh port. Port C is an internal port dedicated to servicing refresh requests, whether they are generated internally by the refresh interval counter, or externally by the user. Two arbitration approaches are available via

Table 6. Refresh Count Interval Table

Ref. Period (μs)	CFS	PLS	FFS	Count Interval CI1, CI0 (8207 Clock Periods)			
				00 (0%)	01 (10%)	10 (20%)	11 (30%)
15.6	1	1	1	236	212	188	164
7.8	1	0	1	118	106	94	82
15.6	1	1	0	148	132	116	100
7.8	1	0	0	74	66	58	50
15.6	0	1	1	118	106	94	82
7.8	0	0	1	59	53	47	41
15.6	0	1	0	74	66	58	50
7.8	0	0	0	37	33	29	25

NOTE:
Refresh period = clock period × refresh count interval.

the Port Priority programming option, program bit PPR. PPR determines whether the most recently used port will remain selected (PPR = 1) or whether Port A will be favored or preferred over Port B (PPR = 0).

A port is selected if the arbiter has given the selected port direct access to the timing generators. The front-end logic, which includes the arbiter, is designed to operate in parallel with the selected port. Thus a request on the selected port is serviced immediately. In contrast, an unselected port only has access to the timing generators through the front-end logic. Before a RAM cycle can start for an unselected port, that port must first become selected (i.e., the MUX output now gates that port's address into the 8207 in the case of Port A or B). Also, in order to allow its address to stabilize, a newly selected port's first RAM cycle is started by the front-end logic. Therefore, the selected port has direct access to the timing generators. What all this means is that a request on a selected port is started immediately, while a request on an unselected port is started two to three clock periods after the request, assuming that the other two ports are idle. Under normal operating conditions, this arbitration time is hidden behind the RAM cycle of the selected port so that as soon as the present cycle is over a new cycle is started. Table 7 lists the arbitration rules for both options.

Port LOCK Function

The LOCK function provides each port with the ability to obtain uninterrupted access to a critical region of memory and, thereby, to guarantee that the opposite port cannot "sneak in" and read from or write to the critical region prematurely.

Only one LOCK pin is present and is multiplexed between the two ports as follows: when MUX is high, the 8207 treats the LOCK input as originating at PORT A, while when MUX is low, the 8207 treats LOCK as originating at PORT B. When the 8207 recognizes a LOCK, the MUX output will remain pointed to the locking port until LOCK is deactivated. Refresh is not affected by LOCK and can occur during a locked memory cycle.

Table 7. The Arbitration Rules for the Most Recently Used Port Priority and for Port A Priority Options Are As Follows:

1.	If only one port requests service, then that port—if not already selected—becomes selected.
2a.	When no service requests are pending, the last selected processor port (Port A or B) will remain selected. (Most Recently Used Port Priority Option.)
2b.	When no service requests are pending, Port A is selected whether it requests service or not. (Port A Priority Option.)
3.	During reset initialization only Port C, the refresh port, is selected.
4.	If no processor requests are pending after reset initialization, Port A will be selected.
5b.	If Ports A and B simultaneously(*) request service while Port C is selected, then the next port to be selected is Port A. (Port A Priority Option.)
6.	If a port simultaneously requests service with the currently selected port, service is granted to the selected port.
7.	The MUX output remains in its last state whenever Port C is selected.
8.	If Port C and either Port A or Port B (or both) simultaneously request service, then service is granted to the requester whose port is already selected. If the selected port is not requesting service, then service is granted to Port C.
9.	If during the servicing of one port, the other port requests service before or simultaneously with the refresh port, the refresh port is selected. A new port is not selected before the presently selected port is deactivated.
10.	Activating LOCK will mask off service requests from Port B if the MUX output is high, or from Port A if the MUX output is low.

NOTE:
*By "simultaneous" it is meant that two or more requests are valid at the clock edge at which the internal arbiter samples them.

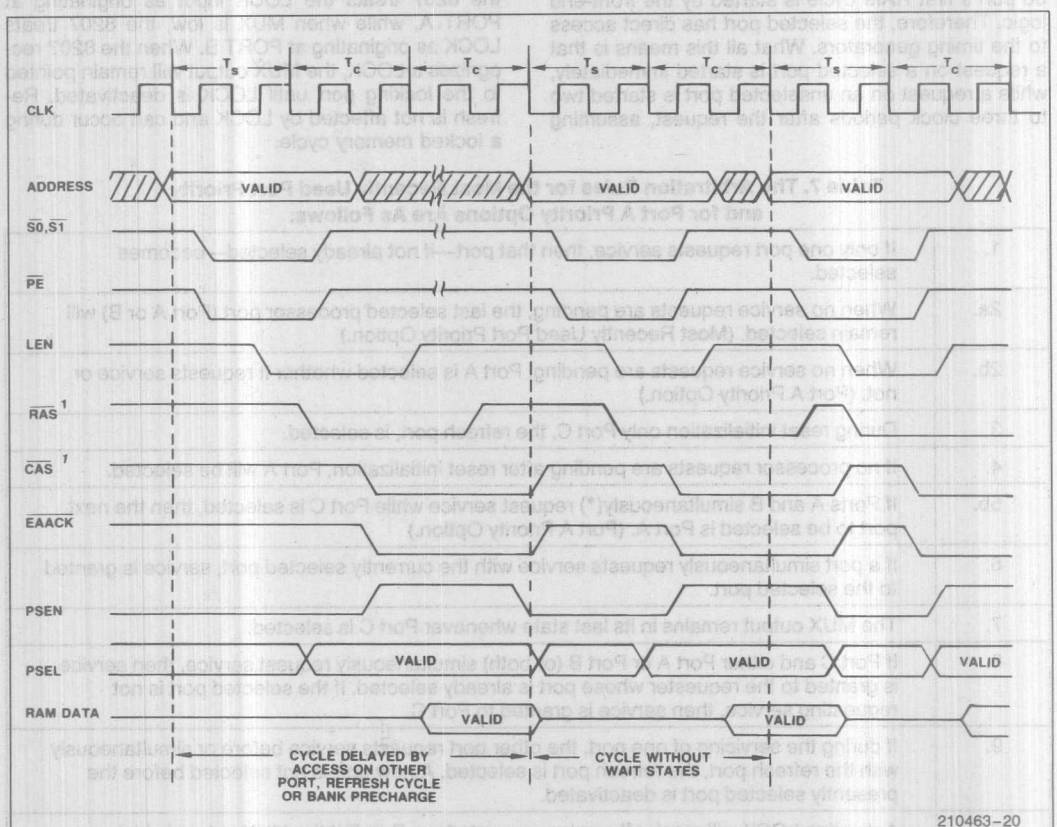
Dual-Port Considerations

For both ports to be operated synchronously, several conditions must be met. The processors must be the same type (Fast or Slow Cycle) as defined by Table 8 and they must have synchronized clocks. Also when processor types are mixed, even though the clocks may be in phase, one frequency may be twice that of the other. So to run both ports synchronous using the status interface, the processors must have related timings (both phase and frequency). If these conditions cannot be met, then one port must run synchronous and the other asynchronous.

Figure 3 illustrates an example of dual-port operation using the processors in the slow cycle group. Note the use of cross-coupled NAND gates at the MUX output for minimizing contention between the two latches, and the use of flip flops on the status lines of the asynchronous processor for delaying the status and thereby guaranteeing RAS will not be issued, even in the worst case, until address is valid.

Processor Timing

In order to run without wait states, $\overline{AACK}$ must be used and connected to the $\overline{SRDY}$ input of the appropriate bus controller. $\overline{AACK}$ is issued relative to a point within the RAM cycle and has no fixed relationship to the processor's request. The timing is such, however, that the processor will run without wait states, barring refresh cycles, bank precharge, and RAM accesses from the other port. In non-ECC fast cycle, fast RAM, non-extended configurations (80286), $\overline{AACK}$ is issued on the next falling edge of



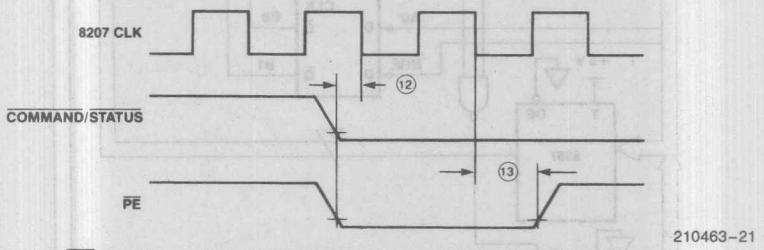
NOTE:
1. The RAS and CAS shown in figure are different banks being accessed.

Figure 14. iAPX 286/8207 Synchronous-Status Timing Programmed in non-ECC Mode, C0 Configuration (Read Cycle)

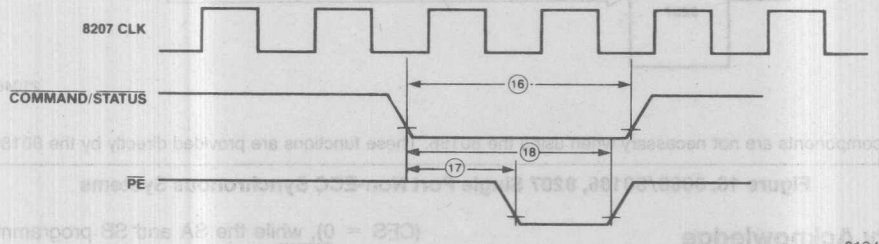
the clock after the edge that issues RAS. In non-ECC, slow cycle, non-extended, or extended with fast RAM cycle configurations (8086, 80188, 80186), $\overline{AACK}$ is issued on the same clock cycle that issues RAS. Figure 14 illustrates the timing relationship between $\overline{AACK}$, the RAM cycle, and the processor cycle for several different situations.

Port Enable ($\overline{PE}$) setup time requirements depend on whether the associated port is configured for synchronous or asynchronous fast or slow cycle operation. In a synchronous fast cycle configuration, $\overline{PE}$ is required to be setup to the same clock edge as the status or commands. If $\overline{PE}$ is true (low), a RAM cycle is started; if not, the cycle is aborted. The memory cycle will only begin when both valid signals ($\overline{PE}$ and $\overline{RD}$ or WR) are recognized at a particular clock edge. In asynchronous operation. $\overline{PE}$ is required to be setup to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command)-to-$\overline{PE}$ delay time, thus allowing for more external decode time that is available in synchronous operation.

The minimum synchronization delay is the additional amount that $\overline{PE}$ must be held valid. If $\overline{PE}$ is not held valid for the maximum synchronization delay time, it is possible that $\overline{PE}$ will go invalid prior to the status or command being synchronized. In such a case the 8207 aborts the cycle. If a memory cycle intended for the 8207 is aborted, then no acknowledge ($\overline{AACK}$ or $\overline{XACK}$) is issued and the processor locks up in endless wait states. Figure 15 illustrates the status (command) timing requirements for synchronous and asynchronous systems. Figures 16 and 17 show a more detailed hook-up of the 8207 to the 8086 and the 80286, respectively.

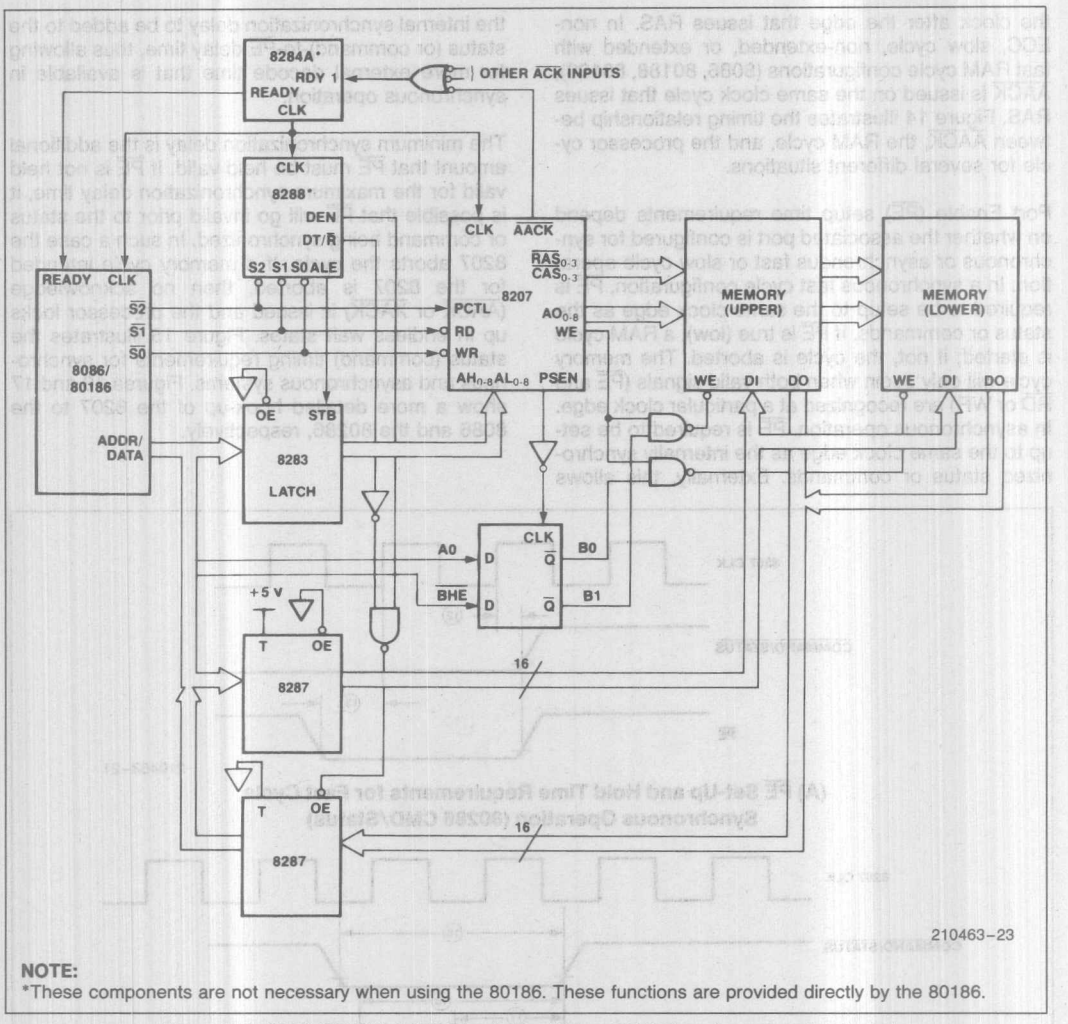


(A) $\overline{PE}$ Set-Up and Hold Time Requirements for Fast Cycle, Synchronous Operation (80286 CMD/Status)



(B) $\overline{PE}$ Timing Requirements for Fast or Slow Cycle Asynchronous Operation

Figure 15



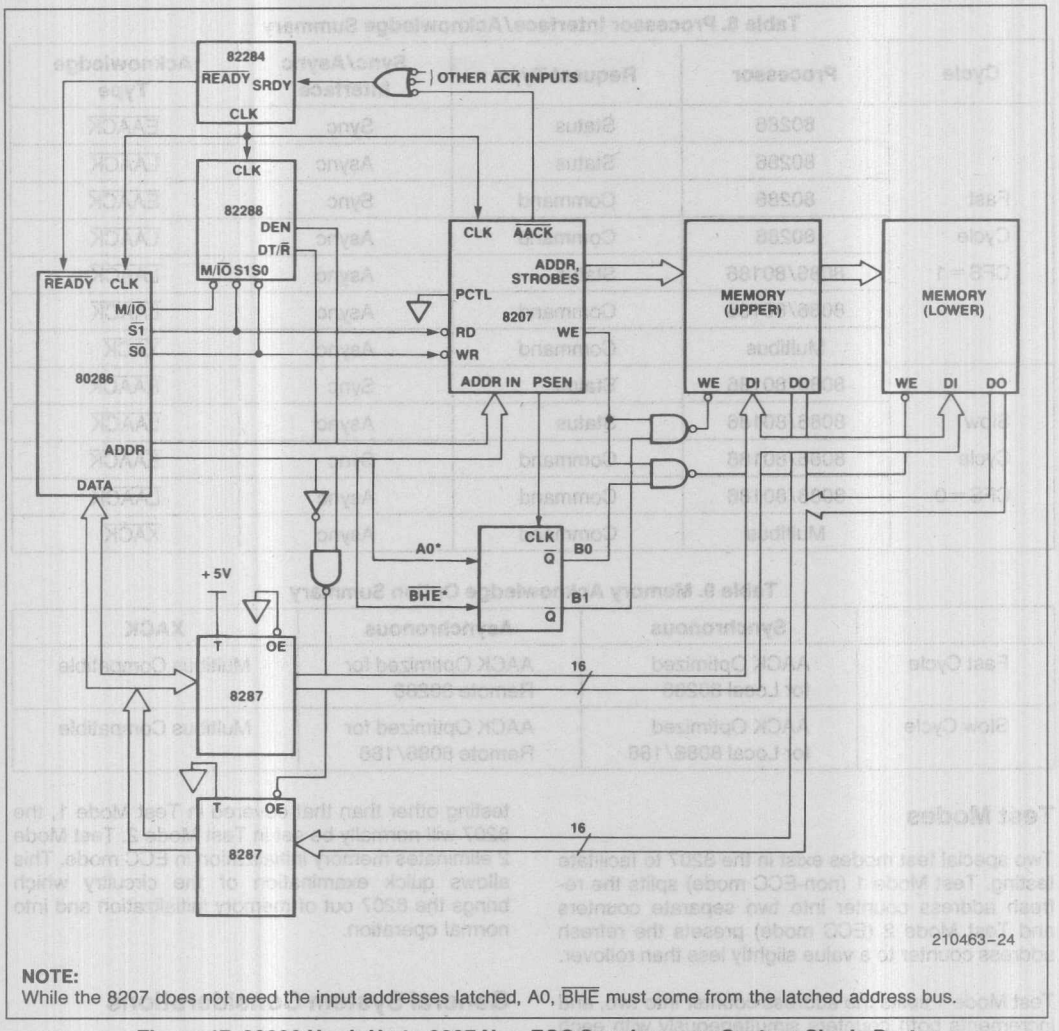
NOTE:
*These components are not necessary when using the 80186. These functions are provided directly by the 80186.

Figure 16. 8086/80186, 8207 Single Port Non-ECC Synchronous Systems

Memory Acknowledge (AACK, XACK)

In system configurations without error correction, two memory acknowledge signals per port are supplied by the 8207. They are the Advanced Acknowledge strobe ($\overline{AACK}$) and the Transfer Acknowledge strobe ($\overline{XACK}$). The CFS programming bit determines for which processor $\overline{AACKA}$ and $\overline{AACKB}$ are optimized, either 80286 (CFS = 1) or 8086/186 (CFS = 0), while the SA and SB programming bits optimize $\overline{AACK}$ for synchronous operation ("early" $\overline{AACK}$) or asynchronous operation ("late" $\overline{AACK}$).

Both the early and late $\overline{AACK}$ strobes are three clocks long for CFS = 1 and two clocks long for CFS = 0. The $\overline{XACK}$ strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the Multibus requirements. $\overline{XACK}$ is removed asynchronously by the command going



NOTE:
While the 8207 does not need the input addresses latched, A0, $\overline{BHE}$ must come from the latched address bus.

Figure 17. 80286 Hook-Up to 8207 Non-ECC Synchronous System-Single Port

inactive. Since in asynchronous operation the 8207 removes read data before late $\overline{AACK}$ or $\overline{XACK}$ is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation, data latching is unnecessary since the 8207 will not remove data until the CPU has read it.

In ECC-based systems there is one memory acknowledge ($\overline{XACK}$ or $\overline{AACK}$) per port and a programming bit associated with each acknowledge. If the X programming bit is active, the strobe is configured as $\overline{XACK}$, while if the bit is inactive, the strobe is configured as $\overline{AACK}$. As in non-ECC, the SA and SB programming bits determine whether the $\overline{AACK}$ strobe is early or late ($\overline{EAACK}$ or $\overline{LAACK}$).

Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Table 9 summarizes the various transfer acknowledge options.

Table 8. Processor Interface/Acknowledge Summary

Cycle	Processor	Request Type	Sync/Async Interface	Acknowledge Type
Fast Cycle CFS=1	80286	Status	Sync	$\overline{\text{EAACK}}$
	80286	Status	Async	$\overline{\text{LAACK}}$
	80286	Command	Sync	$\overline{\text{EAACK}}$
	80286	Command	Async	$\overline{\text{LAACK}}$
	8086/80186	Status	Async	$\overline{\text{LAACK}}$
	8086/80186	Command	Async	$\overline{\text{LAACK}}$
	Multibus	Command	Async	$\overline{\text{XACK}}$
Slow Cycle CFS=0	8086/80186	Status	Sync	$\overline{\text{EAACK}}$
	8086/80186	Status	Async	$\overline{\text{LAACK}}$
	8086/80186	Command	Sync	$\overline{\text{EAACK}}$
	8086/80186	Command	Async	$\overline{\text{LAACK}}$
	Multibus	Command	Async	$\overline{\text{XACK}}$

Table 9. Memory Acknowledge Option Summary

	Synchronous	Asynchronous	XACK
Fast Cycle	AACK Optimized for Local 80286	AACK Optimized for Remote 80286	Multibus Compatible
Slow Cycle	AACK Optimized for Local 8086/186	AACK Optimized for Remote 8086/186	Multibus Compatible

Test Modes

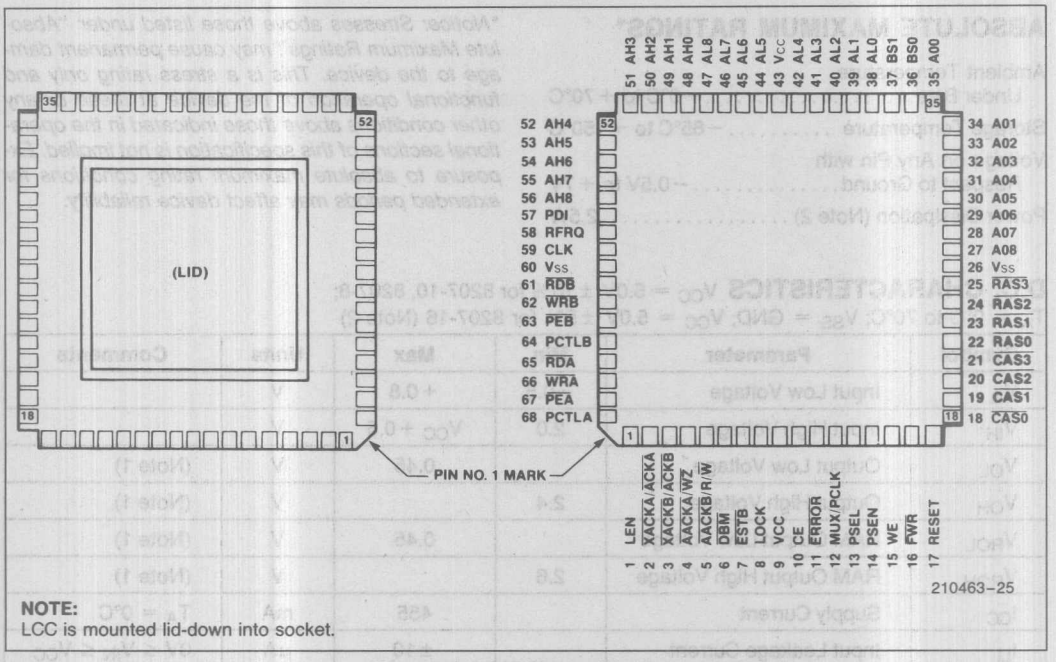
Two special test modes exist in the 8207 to facilitate testing. Test Mode 1 (non-ECC mode) splits the refresh address counter into two separate counters and Test Mode 2 (ECC mode) presets the refresh address counter to a value slightly less than rollover.

Test Mode 1 splits the address counter into two, and increments both counters simultaneously with each refresh address update. By generating external refresh requests, the tester is able to check for proper operation of both counters. Once proper individual counter operation has been established, the 8207 must be returned to normal mode and a second test performed to check that the carry from the first counter increments the second counter. The outputs of the counters are presented on the address out bus with the same timing as the row and column addresses of a normal scrubbing operation. During Test Mode 1, memory initialization is inhibited, since the 8207, be definition, is in non-ECC mode.

Test Mode 2 sets the internal refresh counter to a value slightly less than rollover. During functional testing other than that covered in Test Mode 1, the 8207 will normally be set in Test Mode 2. Test Mode 2 eliminates memory initialization in ECC mode. This allows quick examination of the circuitry which brings the 8207 out of memory initialization and into normal operation.

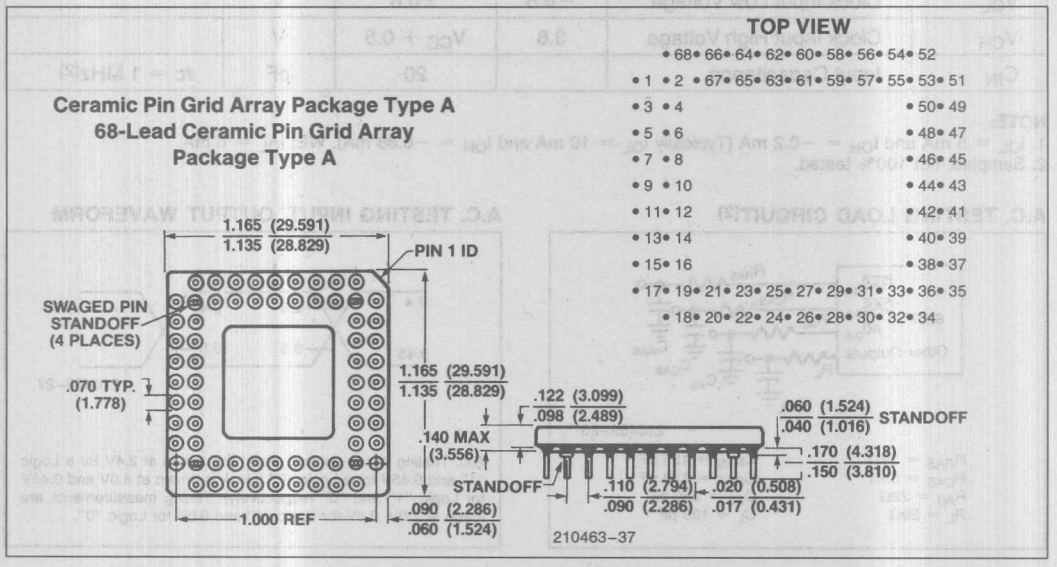
General System Considerations

The RAS_{0-3}, CAS_{0-3}, AO_{0-8}, output buffers were designed to directly drive the heavy capacitive loads associated with dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment and causing noise in other output pins it is necessary to match the output impedance of the RAM output buffers with the RAM array by using series resistors and to add series resistors to other control outputs for noise reduction if necessary. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application. In non-ECC systems unused ECC input pins should be tied high or low to improve noise immunity.



NOTE:
LCC is mounted lid-down into socket.

Figure 19. 8207 Pinout Diagram



8207 Pin Grid Array (PGA) Pin-Out

Packaging

The 8207 is packaged in a 68 lead JEDEC Type A Leadless Chip Carrier (LCC) and in Pin Grid Array (PGA), both in Ceramic. The package designations are R and A respectively.

eg: R 8207-8 LCC, 8 MHz DRAM Controller
eg: A 8207-16 PGA, 16 MHz DRAM Controller

NOTE:
The pin-out of the PGA is the same as the socketed pinout of the LCC.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature
Under Bias ..... −0°C to +70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −0.5V to +7V
Power Dissipation (Note 2) 2.5W

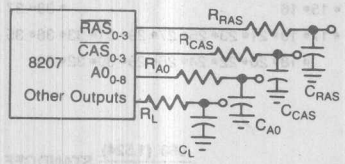
**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS V_{CC} = 5.0V ±10% for 8207-10, 8207-8; T_A = 0°C to 70°C; V_{SS} = GND; V_{CC} = 5.0V ±5% for 8207-16 (Note 2)

Symbol	Parameter	Min	Max	Units	Comments
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 1)
V_{ROL}	RAM Output Low Voltage		0.45	V	(Note 1)
V_{ROH}	RAM Output High Voltage	2.6		V	(Note 1)
I_{CC}	Supply Current		455	mA	T_A = 0°C
I_{LI}	Input Leakage Current		±10	µA	0V ≤ V_{IN} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	±0.6	V	
V_{CH}	Clock Input High Voltage	3.8	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		20	pF	fc = 1 MHz(2)

NOTE:
1. I_{OL} = 5 mA and I_{OH} = −0.2 mA (Typically I_{OL} = 10 mA and I_{OH} = −0.88 mA). WE: I_{OL} = 8 mA.
2. Sampled, not 100% tested.

A.C. TESTING LOAD CIRCUIT(2)

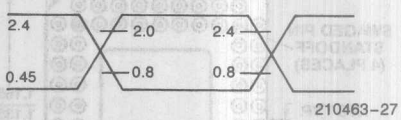


210463-26

R_{RAS} = 39Ω
R_{CAS} = 39Ω
R_{A0} = 22Ω
R_L = 39Ω

C_{RAS} = 150 pF
C_{CAS} = 150 pF
C_{A0} = 380 pF
C_L = 100 pF

A.C. TESTING INPUT, OUTPUT WAVEFORM



210463-27

A.C. Testing inputs (except clock) are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0" (clock is driven at 4.0V and 0.45V for Logic "1" and "0" respectively). Timing measurements are made at 2.0V, 2.4V for Logic "1" and 0.8V for Logic "0".

A.C. CHARACTERISTICS

V_{CC} = 5V ±10% for 8207-8; T_A = 0°C to 70°C; V_{CC} = +5V ±5% for 8207-16

Measurements made with respect to RAS_{0-3}, CAS_{0-3}, AO_{0-8}, are a +2.4V and 0.8V. All other pins are measured at 2.0V and 0.8V. All times are ns unless otherwise indicated. Testing done with specified test load.

Ref	Symbol	Parameter		8207-16, -8		8207-10		Units	Notes
				Min	Max	Min	Max		
CLOCK AND PROGRAMMING									
—	tF	Clock Fall Time			10		10	ns	3
—	tR	Clock Rise Time			10		10	ns	3
1	TCLCL	Clock Period	8207-16	62.5	200			ns	1
			8207-10			100	250	ns	2
			8207-8	125	500			ns	2
2	TCL	Clock Low Time	8207-16	15	180			ns	1
			8207-10			TCLCL/2−12		ns	2
			8207-8	TCLCL/2−12				ns	2
3	TCH	Clock High Time	8207-16	20	180			ns	1
			8207-10			TCLCL/3−3		ns	2
			8207-8	TCLCL/3−3				ns	2
4	TRTVCL	Reset to CLK ↓ Setup		20		40		ns	4
5	TRTH	Reset Pulse Width		4TCLCL		4TCLCL		ns	
6	TPGVRTL	PCTL, PDI, RFRQ to RESET ↓ Setup		125		125		ns	5
7	TRTLPGX	PCTL,RFRQ to RESET ↓ Hold		10		10		ns	
8	TCLPC	PCLK from CLK ↓ Delay			45		45	ns	
9	TPDVCL	PDin to CLK ↓ Setup		60		60		ns	
10	TCLPDX	PDin to CLK ↓ Hold		40		40		ns	6
RAM WARM-UP AND INITIALIZATION									
64	TCLWZL	$\overline{WZ}$ from CLK ↓ Delay			40		40	ns	7
SYNCHRONOUS μP PORT INTERFACE									
11	TPEVCL	$\overline{PE}$ to CLK ↓ Setup		27		27		ns	2
12	TKVCL	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK ↓ Setup		20				ns	1
13	TCLKX	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK ↓ Hold		0		0		ns	
14	TKVCH	$\overline{RD}$, $\overline{WR}$, PCTL to CLK ↑ Setup		20		20		ns	2

A.C. CHARACTERISTICS (Continued)

V_{CC} = 5V ±10% for 8207-8; T_A = 0°C to 70°C; V_{CC} = +5V ±5% for 8207-16

Measurements made with respect to RAS_{0-3}, CAS_{0-3}, AO_{0-8}, are a +2.4V and 0.8V. All other pins are measured at 2.0V and 0.8V. All times are ns unless otherwise indicated. Testing done with specified test load.

Ref	Symbol	Parameter	8207-16, -8		8207-10		Units	Notes
			Min	Max	Min	Max		
ASYNCHRONOUS μP PORT INTERFACE								
15	TRWVCL	$\overline{RD}$, $\overline{WR}$ to CLK ↓ Setup	20		20		ns	8, 9
16	TRWL	$\overline{RD}$, $\overline{WR}$ Pulse Width	2TCLCL+30		2TCLCL+30		ns	
17	TRWLPEV	$\overline{PE}$ from $\overline{RD}$, CFS=1 $\overline{WR}$ ↓ Delay CFS=0		TCLCL−20 TCLCL−30		 TCLCL−20	ns ns	1 2
18	TRWLPEX	$\overline{PE}$ to $\overline{RD}$, $\overline{WR}$ ↓ Hold	2TCLCL+30		2TCLCL+30		ns	
19	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$ ↓ Delay		TCLCL−30		TCLCL−30	ns	2
20	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$ ↓ Hold	2TCLCL+30		2TCLCL+30		ns	2
21	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$ ↓ Delay		2TCLCL−20		2TCLCL−30	ns	1
22	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$ ↓ Hold	3TCLCL+30		3TCLCL+40		ns	1
RAM INTERFACE								
23	TAVCL	AL, AH, BS to CLK ↓ Setup	35+tASR		35+tASR		ns	10
24	TCLAX	AL, AH, BS to CLK ↓ Hold	0		0		ns	
25	TCLLN	LEN from CLK ↓ Delay		35			ns	1
26	TCLRSL	$\overline{RAS}$ ↓ from CLK ↓ Delay		35		35	ns	
27	TRCD	$\overline{RAS}$ to $\overline{CAS}$ CFS=1 Delay CFS=0	TCLCL−25 TCLCL/2−25		 25		ns ns ns	1, 14 11, 14
28	TCLRSH	$\overline{RAS}$ ↑ from CLK ↓ Delay		50		50	ns	
29	TRAH	Row A0 to CFS=1 $\overline{RAS}$ Hold CFS=0	TCLCL/2−11 TCLCL/4−11		 18		ns ns	1, 13, 15 11, 15
30	TASR	Row A0 to $\overline{RAS}$ Setup	0		0			10, 18
31	TASC	Column A0 to CFS=1 $\overline{CAS}$ ↓ Setup CFS=0	0 5		 5		ns ns	13, 19, 20 13, 19, 20

A.C. CHARACTERISTICS (Continued)

V_{CC} = 5V ±10% for 8207-8; T_A = 0°C to 70°C; V_{CC} = +5V ±5% for 8207-16

Measurements made with respect to RAS_{0-3}, CAS_{0-3}, AO_{0-8}, are a +2.4V and 0.8V. All other pins are measured at 2.0V and 0.8V. All times are ns unless otherwise indicated. Testing done with specified test load.

Ref	Symbol	Parameter	8207-16, -8		8207-10		Units	Notes
			Min	Max	Min	Max		
RAM INTERFACE (Continued)								
32	TCAH	Column A0 to $\overline{CAS}$ Hold	(See DRAM Interface Tables)					21
33	TCLCSL	$\overline{CAS}$ ↓ from CLK ↓ Delay	TCLCL/4+30	TCLCL/1.8+53	TCLCL/4+30	100	ns	11, 12
34	TCLCSL	$\overline{CAS}$ ↓ from CLK ↓ Delay		35		40	ns	1
35	TCLCSH	$\overline{CAS}$ ↑ from CLK ↓ Delay		50		50	ns	
36	TCLW	WE from CLK ↓ Delay		35		35	ns	
37	TCLTKL	$\overline{XACK}$ ↓ from CLK ↓ Delay		35		35	ns	
38	TRWLTKH	$\overline{XACK}$ ↑ from $\overline{RD}$ ↑, $\overline{WR}$ ↑ Delay		50		50	ns	
39	TCLAKL	$\overline{AACK}$ ↓ from $\overline{CLK}$ ↓ Delay		35		35	ns	
40	TCLAKH	$\overline{AACK}$ ↑ from CLK ↓ Delay		50		50	ns	
41	TCLDL	$\overline{DBM}$ from CLK ↓ Delay		35		35	ns	
ECC INTERFACE								
42	TWRLFV	$\overline{FWR}$ from CFS=1 $\overline{WR}$ ↓ Delay CFS=0		2TCLCL−40 TCLCL+TCL−40		 100	ns ns	1, 22 2, 22
43	TFVCL	$\overline{FWR}$ to CLK ↓ Setup	40		30		ns	23
44	TCLFX	$\overline{FWR}$ to CLK ↓ Hold	0		0		ns	24
45	TEVCL	$\overline{ERROR}$ to CLK ↓ Setup	20		20		ns	25, 26
46	TCLEX	$\overline{ERROR}$ to CLK ↓ Hold	0		0		ns	
47	TCLRL	R/$\overline{W}$ ↓ from CLK ↓ Delay		40		40	ns	
48	TCLRH	R/$\overline{W}$ ↑ from CLK ↓ Delay		50		50	ns	
49	TCEVCL	CE to CLK ↓ Setup	20		20		ns	25, 27
50	TCLCEX	CE to CLK ↓ Hold	0		0		ns	
51	TCLES	$\overline{ESTB}$ from CLK ↓ Delay		35		45	ns	

A.C. CHARACTERISTICS (Continued)

V_{CC} = 5V ±10% for 8207-8; T_A = 0°C to 70°C; V_{CC} = +5V ±5% for 8207-16

Measurements made with respect to RAS_{0-3}, CAS_{0-3}, AO_{0-8}, are a +2.4V and 0.8V. All other pins are measured at 2.0V and 0.8V. All times are ns unless otherwise indicated. Testing done with specified test load.

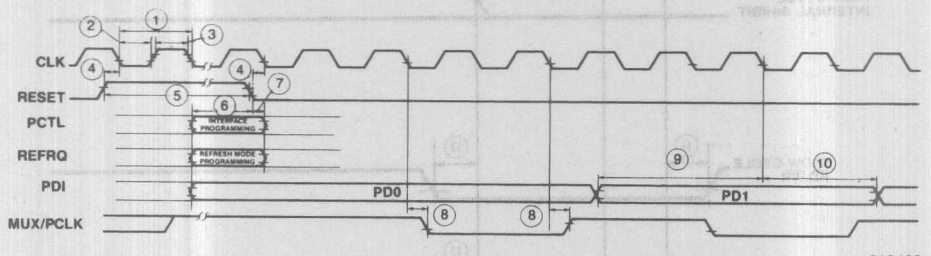
Ref	Symbol	Parameter	8207-16, -8		8207-10		Units	Notes
			Min	Max	Min	Max		
PORT SWITCHING AND LOCK								
52	TCLMV	MUX from CLK ↓ Delay		45		45	ns	
53	TCLPNV	PSEN from CLK ↓ Delay	TCL	TCL+35	TCL	TCL+35	ns	28
54	TCLPSV	PSEL from CLK ↓		35		35	ns	
55	TLKVCL	LOCK to CLK ↓ Setup	30		30		ns	30, 31
56	TCLLKX	LOCK to CLK ↓ Hold	10		10		ns	30, 31
57	TRWLLKV	LOCK from $\overline{RD}$ ↓, $\overline{WR}$ ↓ Delay		2TCLCL−30		2TCLCL−30	ns	31, 32
58	TRWHLKX	LOCK to $\overline{RD}$ ↓, $\overline{WR}$ ↓ Hold	3TCLCL+30		3TCLCL+30		ns	31, 32
REFRESH REQUEST								
59	TRFVCL	RFRQ to CLK ↓ Setup	20		20		ns	
60	TCLRFX	RFRQ to CLK ↓ Hold	10		10		ns	
61	TFRFH	Failsafe RFRQ Pulse Width	TCLCL+30		TCLCL+30		ns	33
62	TRFXCL	Single RFRQ Inactive to CLK ↓ Setup	20		20		ns	34
63	TBRFH	Burst RFRQ Pulse Width	2TCLCL+30		2TCLCL+30		ns	33

NOTES:
1. Specification when programmed in the Fast Cycle processor mode (iAPX 286 mode).
2. Specification when programmed in the Slow Cycle processor mode (iAPX 186 mode).
3. tR and tF are referenced from the 3.5V and 1.0V levels.
4. RESET is internally synchronized to CLK. Hence a set-up time is required only to guarantee its recognition at a particular clock edge.
5. The first programming bit (PD0) is also sampled by RESET going low.
6. TCLPDX is guaranteed if programming data is shifted using PCLK.
7. WZ is issued only in ECC mode.
8. TRWVCL is not required for an asynchronous command except to guarantee its recognition at a particular clock edge.
9. Valid when programmed in either Fast or Slow Cycle mode.
10. tASR is a user specified parameter and its value should be added accordingly to TAVCL.
11. When programmed in Slow Cycle mode and 125 ns ≤ TCLCL < 200 ns.
12. When programmed in Slow Cycle mode and 200 ns ≤ TCLCL.
13. Specification for Test Load conditions.
14. tRCD (actual) = tRCD (specification) + 0.06 (ΔC_{RAS}) − 0.6 (ΔC_{CAS}) where ΔC = C (test load) − C (actual) in pF (These are first order approximations).
15. tRAH (actual) = tRAH (specification) + 0.06 (ΔC_{RAS}) − 0.022 (ΔC_{A0}) where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)
18. tASR (actual) = tASR (specification) + 0.06 (ΔC_{A0}) − 0.025 (ΔC_{RAS}) where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)
19. tASC (actual) = tASC (specification) + 0.06 (ΔC_{A0}) − 0.025 (ΔC_{CAS}) where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)

20. tASC is a function of clock frequency and thus varies with changes in frequency. A minimum value is specified.
21. See 8207 DRAM Interface Tables 14–18.
22. TWRLFV is defined for both synchronous and asynchronous $\overline{FWR}$. In systems in which $\overline{FWR}$ is decoded directly from the address inputs to the 8207, TCLFV is automatically guaranteed by TCLAV.
23. TFVCL is defined for synchronous $\overline{FWR}$.
24. TCLFV is defined for both synchronous and asynchronous $\overline{FWR}$. In systems in which $\overline{FWR}$ is decoded directly from the address inputs to the 8207 TCLFV is automatically guaranteed by TCLAV.
25. $\overline{ERROR}$ and CE are set-up to CLK ↓ in fast cycle mode and CLK ↑ in slow cycle mode.
26. $\overline{ERROR}$ is set-up to the same edge as R/$\overline{W}$ is referenced to, in RMW cycles.
27. CE is set-up to the same edge as WE is referenced to in RMW cycles.
28. Specification when TCL < 25 ns.
29. Synchronous operation only. Must arrive by the second clock falling edge after the clock edge which recognizes the command in order to be effective.
30. LOCK must be held active for the entire period the opposite port must be locked out. One clock after the release of LOCK the opposite port will be able to obtain access to memory.
31. Asynchronous mode only. In this mode a synchronizer stage is used internally in the 8207 to synchronize up LOCK. TRWLLKV and TRWHLKX are only required for guaranteeing that LOCK will be recognized for the requesting port, but these parameters are not required for correct 8207 operation.
32. TFRFH and TBRFH pertain to asynchronous operation only.
33. Single RFRQ cannot be supplied asynchronously.

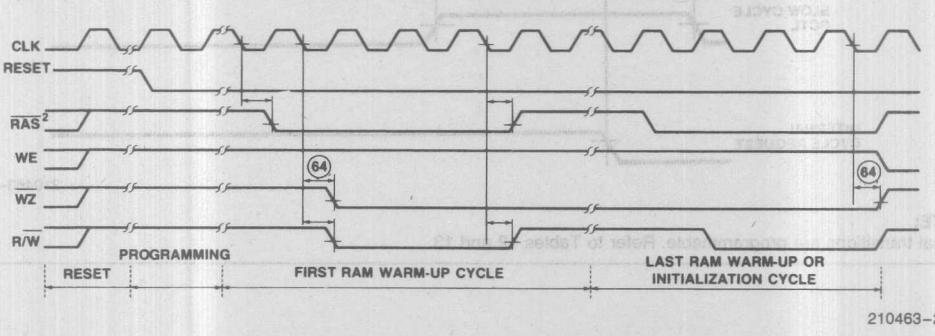
WAVEFORMS

CLOCK AND PROGRAMMING TIMINGS



210463-28

RAM WARM-UP AND MEMORY INITIALIZATION CYCLES

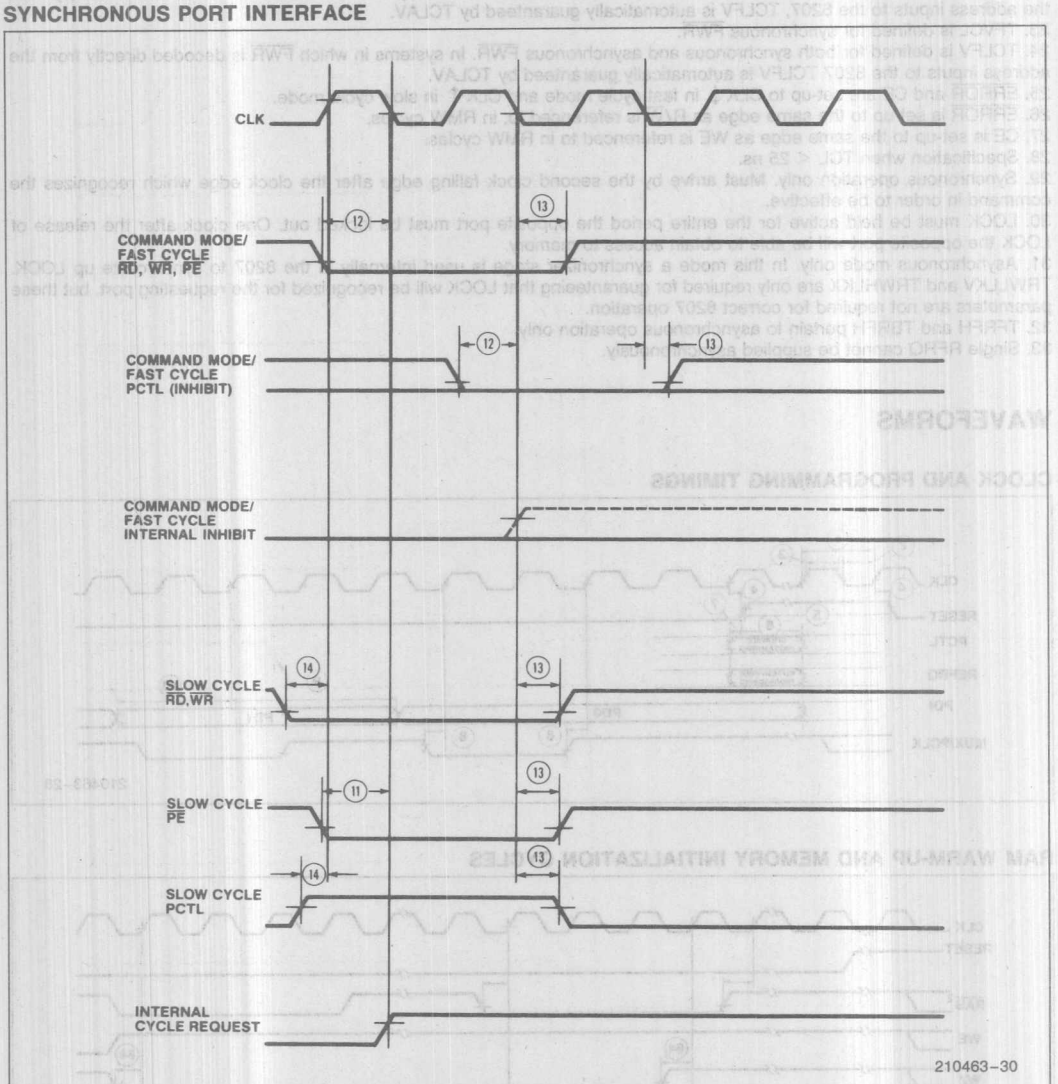


210463-29

NOTES:
1. When in non-ECC mode or in ECC mode with the TM2 programming bit on, there are no initialization cycles, when in ECC mode with TM2 off, the dummy cycles are followed by initialization cycles.
2. The present example assumes a RAS four clocks long.

WAVEFORMS (Continued)

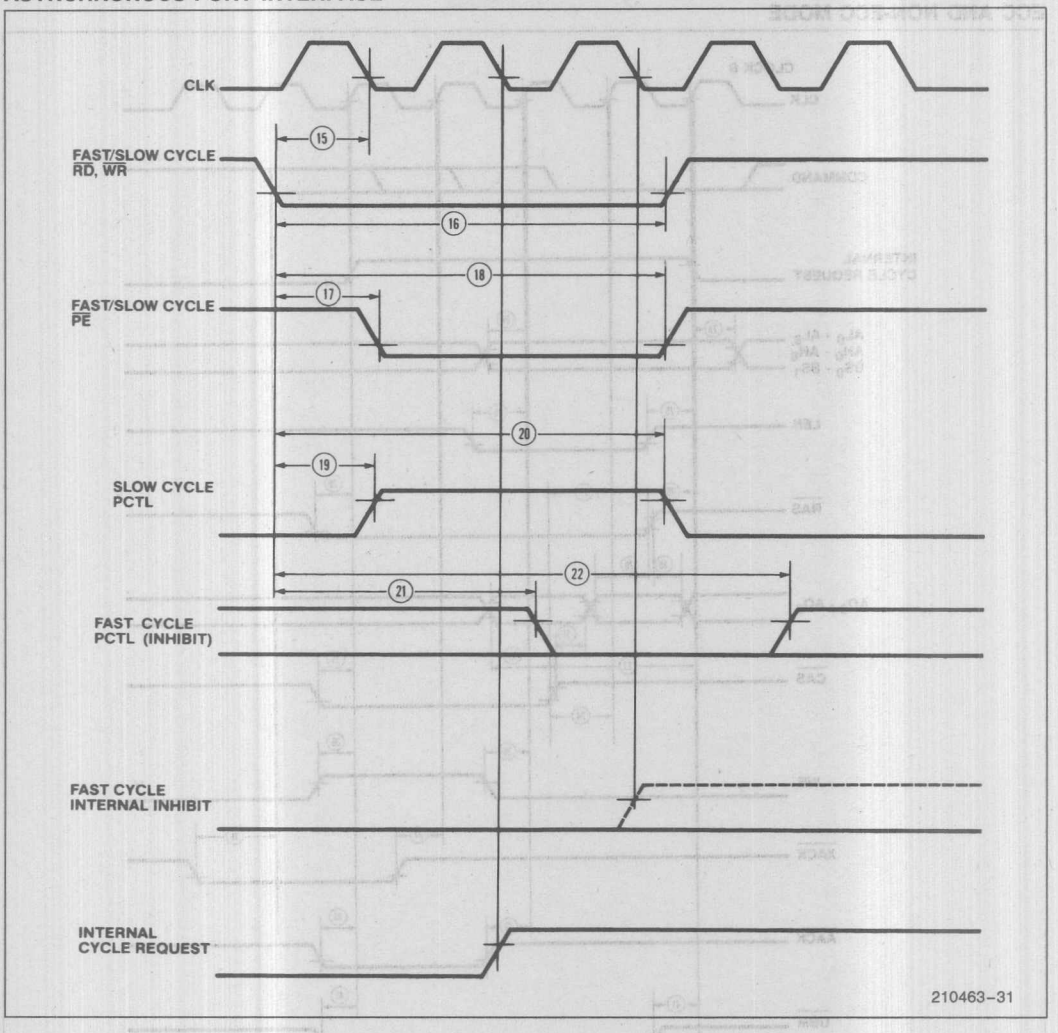
SYNCHRONOUS PORT INTERFACE



NOTE:
Actual transitions are programmable. Refer to Tables 12 and 13.

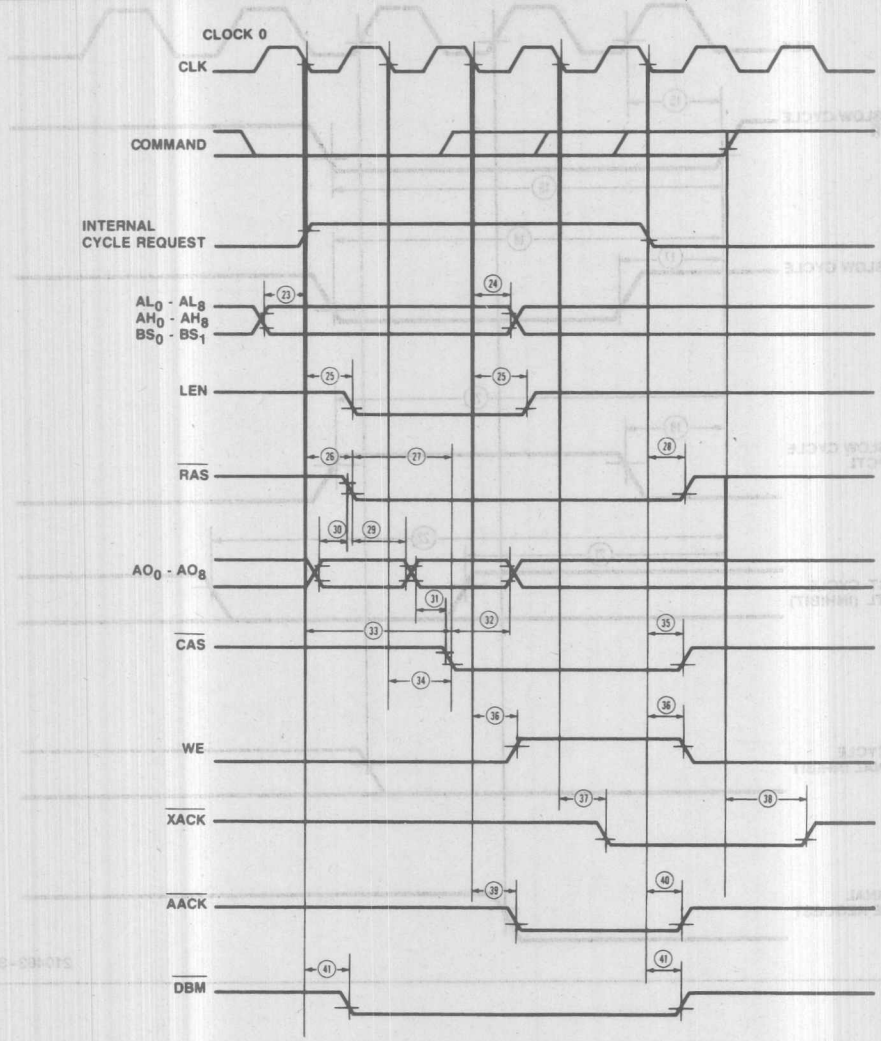
WAVEFORMS (Continued)

ASYNCHRONOUS PORT INTERFACE



WAVEFORMS (Continued)

RAM INTERFACE TIMING
ECC AND NON-ECC MODE

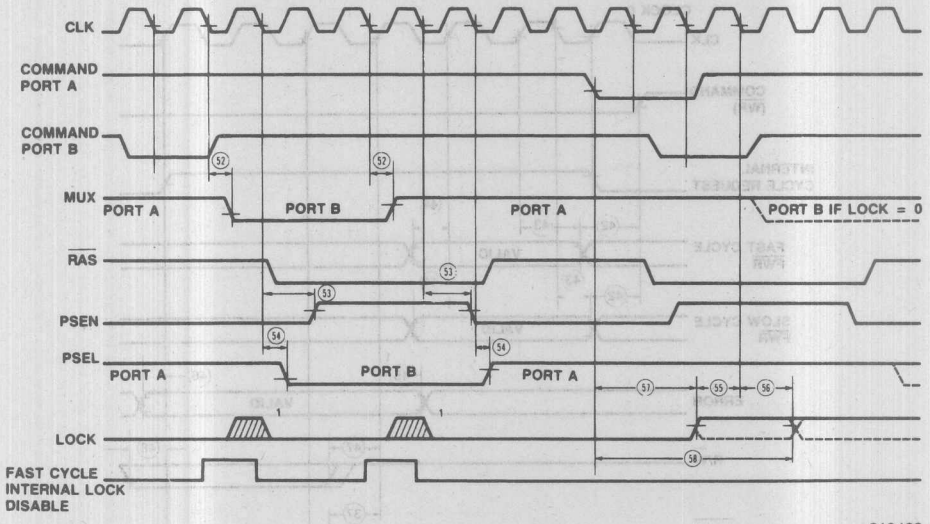


210463-32

NOTE:
Actual transitions are programmable. Refer to Tables 12 and 13.

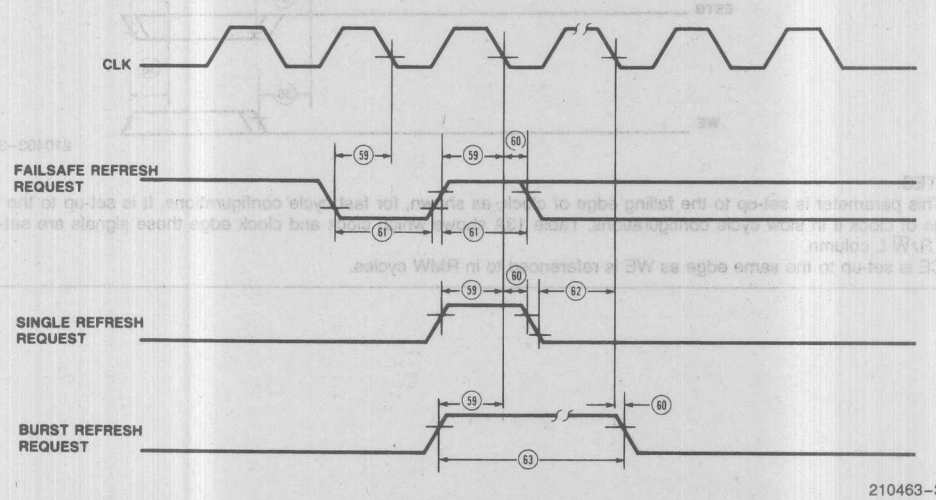
WAVEFORMS (Continued)

PORT SWITCHING AND LOCK TIMING



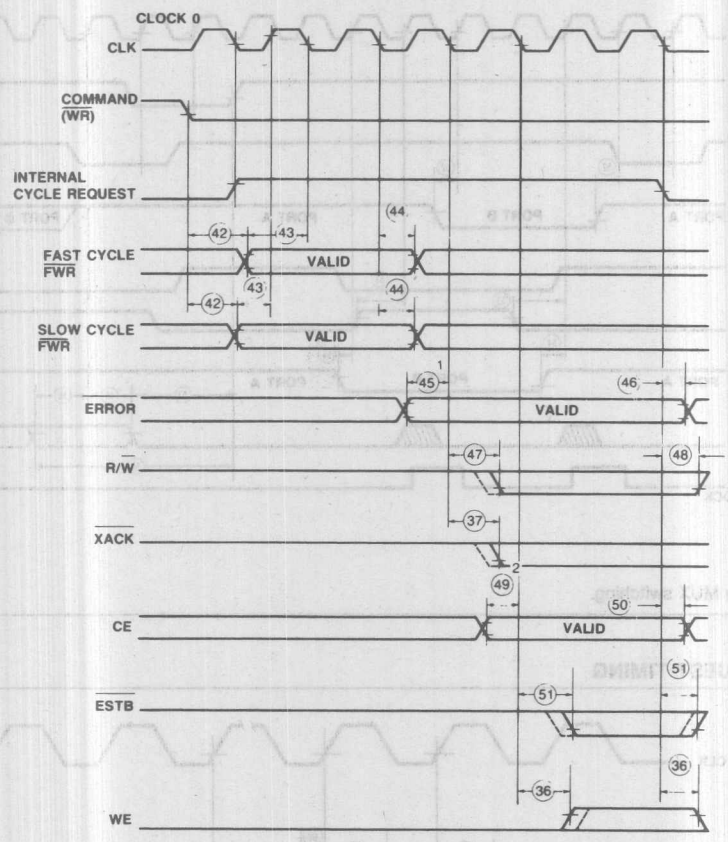
NOTE:
Transients during MUX switching.

REFRESH REQUEST TIMING



WAVEFORMS (Continued)

ECC INTERFACE TIMING



NOTES:
1. This parameter is set-up to the falling edge of clock, as shown, for fast cycle configurations. It is set-up to the rising edge of clock if in slow cycle configurations. Table 13A shows which clock and clock edge these signals are set-up in the R/$\overline{W}$ L column.
2. CE is set-up to the same edge as WE is referenced to in RMW cycles.

CONFIGURATION TIMING CHARTS

The timing charts that follow are based on 8 basic system configurations where the 8207 operates.

Tables 10 and 11 give a description of non-ECC and ECC system configurations based on the 8207's PD0, PD3, PD4, PD10 and PD11 programming bits.

Table 10. Non-ECC System Configurations

Non-ECC Mode: PD0 = 0

Timing Conf.	$\overline{CFS}$(PD3)	$\overline{RFS}$(PD4)	EXT(PD10)	$\overline{FFS}$(PD11)
C_0	iAPX286(0)	Fast RAM(0)	Not EXT(0)	12 MHz(1)
C_0	iAPX286(0)	Fast RAM(0)	EXT(1)	12 MHz(1)
C_0	iAPX286(0)	Slow RAM(1)	Not EXT(0)	12 MHz(1)
C_0	iAPX286(0)	Slow RAM(1)	EXT(1)	12 MHz(1)
C_0	iAPX286(0)	Fast RAM(0)	Not EXT(0)	16 MHz(0)
C_1	iAPX286(0)	Slow RAM(1)	Not EXT(0)	16 MHz(0)
C_1	iAPX286(0)	Fast RAM(0)	EXT(1)	16 MHz(0)
C_2	iAPX286(0)	Slow RAM(1)	EXT(1)	16 MHz(0)
C_3	iAPX186(1)	Fast RAM(0)	Not EXT(0)	10, 8 MHz(0)
C_3	iAPX186(1)	Slow RAM(1)	Not EXT(0)	10, 8 MHz(0)
C_3	iAPX186(1)	Fast RAM(0)	EXT(1)	10, 8 MHz(0)
C_3	iAPX186(1)	Fast RAM(0)	Not EXT(0)	6 MHz(1)
C_3	iAPX186(1)	Fast RAM(0)	EXT(1)	6 MHz(1)
C_3	iAPX186(1)	Slow RAM(1)	Not EXT(0)	6 MHz(1)
C_3	iAPX186(1)	Slow RAM(1)	EXT(1)	6 MHz(1)
C_4	iAPX186(1)	Slow RAM(1)	EXT(1)	10, 8 MHz(0)

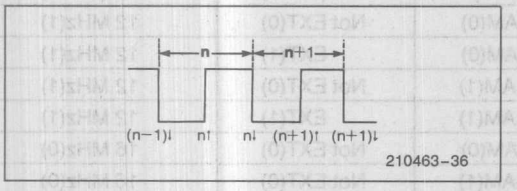
Table 11. ECC System Configurations

ECC Mode: PD0 = 1

Timing Conf.	CFS(PD3)	RFS(PD4)	$\overline{EXT}$(PD10)	FFS(PD11)
C_0	iAPX286(1)	Slow RAM(0)	M/S EDCU(0)	12 MHz(0)
C_0	iAPX286(1)	Slow RAM(0)	M EDCU(1)	12 MHz(0)
C_0	iAPX286(1)	Fast RAM(1)	M/S EDCU(0)	12 MHz(0)
C_0	iAPX286(1)	Fast RAM(1)	M EDCU(1)	12 MHz(0)
C_0	iAPX286(1)	Fast RAM(1)	M EDCU(1)	16 MHz(1)
C_1	iAPX286(1)	Slow RAM(0)	M EDCU(1)	16 MHz(1)
C_2	iAPX286(1)	Fast RAM(1)	M/S EDCU(0)	16 MHz(1)
C_3	iAPX286(1)	Slow RAM(0)	M/S EDCU(0)	16 MHz(1)
C_4	iAPX186(0)	Slow RAM(0)	M/S EDCU(0)	6 MHz(0)
C_4	iAPX186(0)	Fast RAM(1)	M/S EDCU(0)	6 MHz(0)
C_4	iAPX186(0)	Slow RAM(0)	M EDCU(1)	10, 8 MHz(1)
C_4	iAPX186(0)	Fast RAM(1)	M EDCU(1)	10, 8 MHz(1)
C_5	iAPX186(0)	Slow RAM(0)	M/S EDCU(0)	10, 8 MHz(1)
C_5	iAPX186(0)	Fast RAM(1)	M/S EDCU(0)	10, 8 MHz(1)
C_6	iAPX186(0)	Slow RAM(0)	M EDCU(1)	6 MHz(0)
C_6	iAPX186(0)	Fast RAM(1)	M EDCU(1)	6 MHz(0)

Using the Timing Charts

The notation used to indicate which clock edge triggers an output transition is "n↑" or "n↓", where "n" is the number of clock periods that have passed since clock 0, the reference clock, and "↑" refers to rising edge and "↓" to falling edge. A clock period is defined as the interval from a clock falling edge to the following falling edge. Clock edges are defined as shown below.



The clock edges which trigger transitions on each 8207 output are tabulated in Table 12 for non-ECC mode and Table 13 for ECC mode. "H" refers to the high-going transition, and "L" to low-going transition; "V" refers to valid, and "$\overline{V}$" to non-valid.

Clock 0 is defined as the clock in which the 8207 begins a memory cycle, either as a result of a port request which has just arrived, or of a port request which was stored previously but could not be serviced at the time of its arrival because the 8207 was performing another memory cycle. Clock 0 may be identified externally by the leading edge of RAS, which is always triggered on 0↓.

Notes for interpreting the timing charts:

1. **PSEL - valid** is given as the latest time it can occur. It is entirely possible for PSEL to become valid before the time given in a refresh cycle. PSEL can switch as defined in the chart, but it has no bearing on the refresh cycle itself, but only on a subsequent cycle for one of the external ports.
2. **LEN - low** is given as the latest time it can occur. LEN is only activated by port A configured in Fast Cycle iAPX286 mode, and thus it is not activated by a refresh cycle, although it may be activated by port A during a refresh cycle.
3. **ADDRESS - col** is the time column address becomes valid.
4. In non-ECC mode the $\overline{CAS}$, $\overline{EAACK}$, $\overline{LAACK}$ and $\overline{XACK}$ outputs are not issued during refresh.
5. In ECC mode there are really seven types of cycles: Read without error, read with error, full write, partial write without error, partial write with error, refresh without error, and refresh with error. These cycles may be derived from the timing chart as follows:
   A. Read without error: Use row marked 'RD, RF'.
   B. Read with error: Use row marked 'RMW', except for $\overline{EAACK}$ and $\overline{LAACK}$, which should be taken from 'RD, RF'. If the error is uncorrectable. WE will not be issued.
   C. Full write: Use row marked 'WR'.
   D. Partial write without error: Use row marked 'RMW', except that $\overline{DBM}$ and $\overline{ESTB}$ will not be issued.
   E. Partial write with error: Use row marked 'RMW', except that $\overline{DBM}$ will not be issued. If the error is uncorrectable, WE will not be issued.
   F. Refresh without error: Use row marked 'RD, RF', except that $\overline{ESTB}$, $\overline{EAACK}$, $\overline{LAACK}$, and $\overline{XACK}$ will not be issued.
   G. Refresh with error: Use row marked 'RMW', except that $\overline{EAACK}$, $\overline{LAACK}$, $\overline{ESTB}$, and $\overline{XACK}$ will not be issued. If the error is uncorrectable WE will not be issued.
6. **XACK - high** is reset asynchronously by command going inactive and not by a clock edge.
7. **MUX - valid** is given as the latest time it can occur.

Table 12A. Timing Chart—Non-ECC Mode

		PSEN		PSEL		$\overline{DBM}$		LEN		$\overline{RAS}$		$\overline{CAS}$		WE	
C_n	Cycle	H	L	V	$\overline{V}$	L	H	L	H	L	H	L	H	H	L
C_0	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	3↓	1↓	4↓		
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	1↓	5↓	2↓	5↓
C_1	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓		
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	1↓	5↓	2↓	5↓
C_2	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓		
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	1↓	5↓	2↓	5↓
C_3	RD, RF	0↓	2↓	0↓	3↓	0↓	3↓			0↓	3↓	0↓	3↓		
	WR	0↓	3↓	0↓	4↓					0↓	4↓	0↓	4↓	2↑	4↓
C_4	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓			0↓	4↓	0↓	4↓		
	WR	0↓	3↓	0↓	4↓					0↓	4↓	0↓	4↓	2↑	4↓

Table 12B. Timing Chart—Non-ECC Mode

		Col Addr		$\overline{EAACK}$		$\overline{LAACK}$		$\overline{XACK}$		MUX	
C_n	Cycle	V	$\overline{V}$	L	H	L	H	L	H	V	$\overline{V}$
C_0	RD, RF	0↓	2↓	1↓	4↓	2↓	5↓	3↓	$\overline{RD}$	−2↓	2↓
	WR	0↓	2↓	1↓	4↓	1↓	4↓	3↓	$\overline{WR}$	−2↓	2↓
C_1	RD, RF	0↓	3↓	2↓	5↓	2↓	5↓	4↓	$\overline{RD}$	−2↓	2↓
	WR	0↓	3↓	1↓	4↓	1↓	4↓	3↓	$\overline{WR}$	−2↓	2↓
C_2	RD, RF	0↓	3↓	2↓	5↓	3↓	6↓	4↓	$\overline{RD}$	−2↓	2↓
	WR	0↓	3↓	1↓	4↓	1↓	4↓	3↓	$\overline{WR}$	−2↓	2↓
C_3	RD, RF	0↓	2↓	0↓	2↓	1↓	3↓	2↓	$\overline{RD}$	−1↓	2↓
	WR	0↓	2↓	0↓	2↓	1↑	3↑	2↓	$\overline{WR}$	−1↓	2↓
C_4	RD, RF	0↓	2↓	1↓	3↓	1↓	3↓	3↑	$\overline{RD}$	−1↓	2↓
	WR	0↓	2↓	0↓	2↓	1↑	3↑	2↓	$\overline{WR}$	−1↓	2↓

Table 13A. Timing Chart—ECC Mode

		PSEN		PSEL		$\overline{DBM}$		$\overline{LEN}$		$\overline{RAS}$		$\overline{CAS}$		R/$\overline{W}$		WE	
C_n	Cycle	H	L	V	$\overline{V}$	L	H	L	H	L	H	L	H	L	H	H	L
C_0	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓				
	WR	0↓	5↓	0↓	6↓			0↓	2↓	0↓	6↓	1↓	6↓	1↓	6↓	3↓	6↓
	RMW	0↓	8↓	0↓	9↓	0↓	9↓	0↓	2↓	0↓	9↓	1↓	9↓	4↓	9↓	6↓	9↓
C_1	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓				
	WR	0↓	5↓	0↓	6↓			0↓	2↓	0↓	6↓	1↓	6↓	1↓	6↓	3↓	6↓
	RMW	0↓	8↓	0↓	9↓	0↓	9↓	0↓	2↓	0↓	9↓	1↓	9↓	4↓	9↓	6↓	9↓
C_2	RD, RF	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	5↓	1↓	7↓				
	WR	0↓	6↓	0↓	7↓			0↓	2↓	0↓	7↓	1↓	7↓	1↓	7↓	4↓	7↓
	RMW	0↓	10↓	0↓	11↓	0↓	11↓	0↓	2↓	0↓	11↓	1↓	11↓	5↓	11↓	8↓	11↓
C_3	RD, RF	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	5↓	1↓	7↓				
	WR	0↓	6↓	0↓	7↓			0↓	2↓	0↓	7↓	1↓	7↓	1↓	7↓	4↓	7↓
	RMW	0↓	10↓	0↓	11↓	0↓	11↓	0↓	2↓	0↓	11↓	1↓	11↓	5↓	11↓	8↓	11↓
C_4	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓			0↓	3↓	0↓	4↓				
	WR	0↓	4↓	0↓	5↓					0↓	5↓	0↓	5↓	1↑	5↓	3↑	5↓
	RMW	0↓	6↓	0↓	7↓	0↓	7↓			0↓	7↓	0↓	7↓	3↑	7↓	5↑	7↓
C_5	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓			0↓	3↓	0↓	4↓				
	WR	0↓	4↓	0↓	5↓					0↓	5↓	0↓	5↓	1↑	5↓	3↑	5↓
	RMW	0↓	6↓	0↓	7↓	0↓	7↓			0↓	7↓	0↓	7↓	3↑	7↓	5↑	7↓
C_6	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓			0↓	3↓	0↓	4↓				
	WR	0↓	3↓	0↓	4↓					0↓	4↓	0↓	4↓	1↑	4↓	2↑	4↓
	RMW	0↓	4↓	0↓	5↓	0↓	5↓			0↓	5↓	0↓	5↓	2↑	5↓	3↑	5↓

Table 13B. Timing Chart—ECC Mode

		Col Addr		$\overline{ESTB}$		$\overline{EAACK}$		$\overline{LAACK}$		$\overline{XACK}$		$\overline{MUX}$	
C_n	Cycle	V	$\overline{V}$	L	H	L	H	L	H	L	H	V	$\overline{V}$
C_0	RD, RF	0↓	2↓			2↓	5↓	3↓	6↓	4↓	RD	−2↓	2↓
	WR	0↓	2↓			2↓	5↓	2↓	5↓	4↓	WR	−2↓	2↓
	RMW	0↓	2↓	6↓	8↓	5↓	8↓	5↓	8↓	7↓	WR	−2↓	2↓
C_1	RD, RF	0↓	3↓			3↓	6↓	3↓	6↓	4↓	RD	−2↓	2↓
	WR	0↓	3↓			2↓	5↓	2↓	5↓	4↓	WR	−2↓	2↓
	RMW	0↓	3↓	6↓	8↓	5↓	8↓	5↓	8↓	7↓	WR	−2↓	2↓
C_2	RD, RF	0↓	3↓			4↓	7↓	4↓	7↓	5↓	RD	−2↓	2↓
	WR	0↓	3↓			3↓	6↓	3↓	6↓	5↓	WR	−2↓	2↓
	RMW	0↓	3↓	8↓	10↓	7↓	10↓	7↓	10↓	9↓	WR	−2↓	2↓
C_3	RD, RF	0↓	3↓			4↓	7↓	5↓	8↓	5↓	RD	−2↓	2↓
	WR	0↓	3↓			3↓	6↓	3↓	6↓	5↓	WR	−2↓	2↓
	RMW	0↓	3↓	8↓	10↓	7↓	10↓	7↓	10↓	9↓	WR	−2↓	2↓
C_4	RD, RF	0↓	2↓			1↓	3↓	2↑	4↑	3↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓
	RMW	0↓	2↓	5↑	6↑	3↓	5↓	4↑	6↑	5↓	WR	−1↓	2↓
C_5	RD, RF	0↓	2↓			2↓	4↓	3↑	5↑	3↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓
	RMW	0↓	2↓	5↑	6↑	3↓	5↓	4↑	6↑	5↓	WR	−1↓	2↓
C_2	RD, RF	0↓	2↓			1↓	3↓	1↑	3↑	2↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	1↑	3↑	2↓	WR	−1↓	2↓
	RMW	0↓	2↓	3↑	4↑	1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓

8207—DRAM Interface Parameter Equations

Several DRAM parameters, but not all, are a direct function of 8207 timings, and the equations for these parameters are given in the following tables. The following is a list of those DRAM parameters which have NOT been included in the following tables, with an explanation for their exclusion.

READ, WRITE, READ-MODIFY-WRITE & REFRESH CYCLES

tRAC: response parameter.

tCAC: response parameter.

tREF: See "Refresh Period Options"

tCRP: must be met only if $\overline{CAS}$-only cycles, which do not occur with 8207, exist.

tRAH: See "A.C. Characteristics"

tRCD: See "A.C. Characteristics"

tASC: See "A.C. Characteristics"

tASR: See "A.C. Characteristics"

tOFF: response parameter.

READ & REFRESH CYCLES

tRCH: WE always goes active after $\overline{CAS}$ goes active, hence tRCH is guaranteed by tCPN.

WRITE CYCLE

tRC: guaranteed by tRWC.

tRAS: guaranteed by tRRW.

tCAS: guaranteed by tCRW.

tWCS: WE always activated after $\overline{CAS}$ is activated, except in memory initialization, hence tWCS is always negative (this is important for RMW only) except in memory initialization; in memory initialization tWCS is positive and has several clocks of margin.

tDS: system-dependent parameter.

tDH: system-dependent parameter.

tDHR: system-dependent parameter.

READ-MODIFY-WRITE CYCLE

tRWD: don't care in 8207 write cycles, but tabulated for 8207 RMW cycles.

tCWD: don't care in 8207 write cycles, but tabulated for 8207 RMW cycles.

Table 14. Non-ECC Mode—RD, RF Cycles

Parameter	Fast Cycle Configurations			Slow Cycle Configurations		Notes
	C_0	C_1	C_2	C_3	C_4	
tRP	3TCLCL−T26	4TCLCL−T26	4TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	3TCLCL−T35	3TCLCL−T35	3TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	2TCLCL−T34	3TCLCL−T34	3TCLCL−T34	3TCLCL−T34	4TCLCL−T34	1
tCSH	4TCLCL−T26	6TCLCL−T26	6TCLCL−T26	3TCLCL−T26	4TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	3/30	2
tRC	6TCLCL	8TCLCL	8TCLCL	5TCLCL	6TCLCL	1
tRAS	3TCLCL−T26	4TCLCL−T26	4TCLCL−T26	3TCLCL−T26	4TCLCL−T26	1
tCAS	3TCLCL−T34	5TCLCL−T34	5TCLCL−T34	3TCLCL−T34	4TCLCL−T34	1
tRCS	2TCLCL−TCL −T36−TBUF	2TCLCL−TCL −T36−TBUF	2TCLCL−TCL −T36−TBUF	1.5TCLCL−TCL −T36−TBUF	1.5TCLCL−TCL −T36−TBUF	1

Table 15. Non-ECC Mode—WR Cycle

Parameter	Fast Cycle Configurations			Slow Cycle Configurations		Notes
	C_0	C_1	C_2	C_3	C_4	
tRP	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tCSH	5TCLCL−T26	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	3/30	2
tRWC	8TCLCL	8TCLCL	8TCLCL	6TCLCL	6TCLCL	1
tRRW	5TCLCL−T26	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCRW	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tWCH	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	1, 3
tWCR	4TCLCL+TCL −T26	4TCLCL+TCL −T26	4TCLCL+TCL −T26	3TCLCL+TCL −T26	3TCLCL+TCL −T26	1, 3
tWP	2TCLCL+TCL −T36−TBUF	2TCLCL+TCL −T36−TBUF	2TCLCL+TCL −T36−TBUF	2TCLCL−T36 −TBUF	2TCLCL−T36 −TBUF	1
tRWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tCWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1

Table 16A. ECC Mode—RD, RF Cycles

Parameter	Fast Cycle Mode				Notes
	C_0	C_1	C_2	C_3	
tRP	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCPN	3TCLCL−T35	3TCLCL−T35	3TCLCL−T35	3TCLCL−T35	1
tRSH	3TCLCL−T34	3TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tCSH	6TCLCL−T26	6TCLCL−T26	7TCLCL−T26	7TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	2
tRC	8TCLCL	8TCLCL	9TCLCL	9TCLCL	1
tRAS	4TCLCL−T26	4TCLCL−T26	5TCLCL−T26	5TCLCL−T26	1
tCAS	5TCLCL−T34	5TCLCL−T34	6TCLCL−T34	6TCLCL−T34	1
tRCS	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	1

Table 16B. ECC Mode—RD, RF Cycles

Parameter	Slow Cycle Mode			Notes
	C_4	C_5	C_6	
tRP	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	1.5TCLCL−T35	1.5TCLCL−T35	1.5TCLCL−T35	1
tRSH	3TCLCL−T34	3TCLCL−T34	3TCLCL−T34	1
tCSH	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCAH	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	2
tRC	5TCLCL	5TCLCL	5TCLCL	1
tRAS	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tCAS	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tRCS	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	1

Table 17A. ECC Mode—WR Cycle

Parameter	Fast Cycle Mode				Notes
	C_0	C_1	C_2	C_3	
tRP	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tCPN	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	1
tRSH	5TCLCL−T34	5TCLCL−T34	6TCLCL−T34	6TCLCL−T34	1
tCSH	6TCLCL−T26	6TCLCL−T26	7TCLCL−T26	7TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	2
tRWC	9TCLCL	9TCLCL	10TCLCL	10TCLCL	1
tRRW	6TCLCL−T26	6TCLCL−T26	7TCLCL−T26	7TCLCL−T26	1
tCRW	5TCLCL−T34	5TCLCL−T34	6TCLCL−T34	6TCLCL−T34	1
tWCH	5TCLCL−T34	5TCLCL−T34	6TCLCL−T34	6TCLCL−T34	1, 4
tWCR	6TCLCL−T26	6TCLCL−T26	7TCLCL−T26	7TCLCL−T26	1, 4
tWP	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1
tRWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1
tCWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1

Table 17B. ECC Mode—WR Cycle

Parameter	Slow Cycle Mode			Notes
	C_4	C_5	C_6	
tRP	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	2.5TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	5TCLCL−T34	5TCLCL−T34	4TCLCL−T34	1
tCSH	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	1
tCAH	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	2
tRWC	7TCLCL	7TCLCL	6TCLCL	1
tRRW	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	1
tCRW	5TCLCL−T34	5TCLCL−T34	4TCLCL−T34	1
tWCH	5TCLCL−T34	5TCLCL−T34	4TCLCL−T34	1, 4
tWCR	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	1, 4
tWP	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tRWL	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tCWL	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1

Table 18A. ECC Mode—RMW

Parameter	Fast Cycle Mode				Notes
	C_0	C_1	C_2	C_3	
tRP	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tCPN	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	1
tRSH	8TCLCL−T34	8TCLCL−T34	10TCLCL−T34	10TCLCL−T34	1
tCSH	9TCLCL−T26	9TCLCL−T26	11TCLCL−T26	11TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	2
tRWC	12TCLCL	12TCLCL	14TCLCL	14TCLCL	1
tRRW	9TCLCL−T26	9TCLCL−T26	11TCLCL−T26	11TCLCL−T26	1
tCRW	8TCLCL−T34	8TCLCL−T34	10TCLCL−T34	10TCLCL−T34	1
tRCS	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	1
tRWD	6TCLCL−T26	6TCLCL−T26	8TCLCL−T26	8TCLCL−T26	1, 4
tCWD	5TCLCL−T34	5TCLCL−T34	7TCLCL−T34	7TCLCL−T34	1
tWP	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1
tRWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1
tCWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1

Table 18B. ECC Mode—RMW

Parameter	Slow Cycle Mode			Notes
	C_4	C_5	C_6	
tRP	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	2.5TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	7TCLCL−T34	7TCLCL−T34	5TCLCL−T34	1
tCSH	7TCLCL−T26	7TCLCL−T26	5TCLCL−T26	1
tCAH	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	2
tRWC	9TCLCL	9TCLCL	7TCLCL	1
tRRW	7TCLCL−T26	7TCLCL−T26	5TCLCL−T26	1
tCRW	7TCLCL−T34	7TCLCL−T34	5TCLCL−T34	1
tRCS	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	1
tRWD	4TCLCL+TCL −T26	4TCLCL+TCL −T26	2TCLCL+TCL −T26	1
tCWD	4TCLCL+TCL −T34	4TCLCL+TCL −T34	2TCLCL+TCL −T34	1
tWP	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tRWL	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tCWL	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1

NOTES:
1. Minimum.
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization only.
4. Applies to the eight warm-up cycles and to the memory initialization cycles during initialization only.
5. TP = TCLCL
   T26 = TCLRSL
   T34 = TCLCSL
   T35 = TCLCSH
   T36 = TCLW
   TBUF = TTL Buffer delay.

82C08
CHMOS DYNAMIC RAM CONTROLLER

- 0 Wait State with INTEL μProcessors
- iAPX 286 (10, 8 MHz) } 82C08-20 20 MHz, 82C08-16 16 MHz
  iAPX 186/88 86/88 } 82C08-10 10 MHz, 82C08-8 8 MHz
- Supports 64K and 256K DRAMs (256K x 1 and 256K x 4 Organizations)
- Power Down Mode with Programmable Memory Refresh using Battery Backup
- Directly Addresses and Drives up to 1 Megabyte without External Drivers
- Microprocessor Data Transfer and Advance Acknowledge Signals
- Five Programmable Refresh Modes
- Automatic RAM Warm-up
- Pin-Compatible with 8208
- 48 Lead Plastic DIP; 68 Lead PLCC
  (See Intel Packaging; Order Number: 231369-001)
- Compatible with Normal Modes of Static Column and Ripplemode DRAMs

The Intel 82C08 Dynamic RAM Controller is a CMOS, high performance, systems oriented, Dynamic RAM controller that is designed to easily interface 64K and 256K Dynamic RAMs to Intel and other microprocessors. The 82C08 also has a power down mode where only the refresh logic is activated using battery backup.

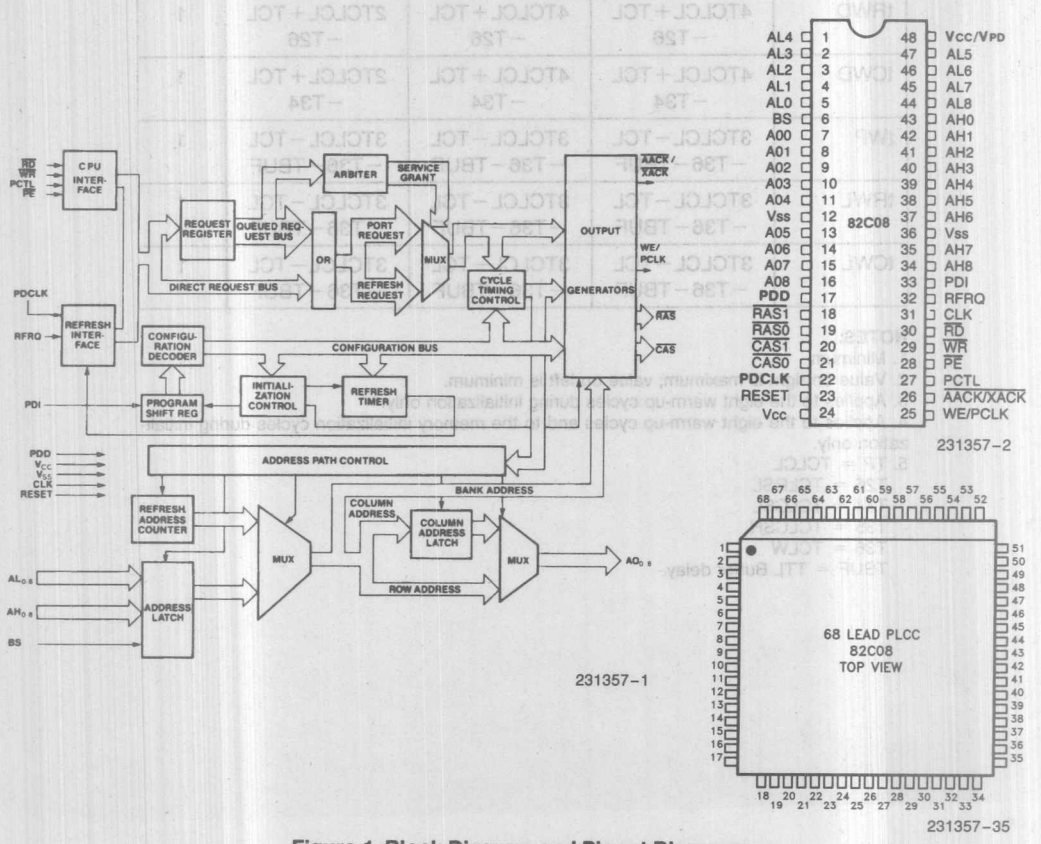


Figure 1. Block Diagram and Pinout Diagrams

September 1987
Order Number: 231357-006

Table 1. Pin Description

Symbol	DIP Pin	PLCC	Type	Name and Function
AL0	5	55	I	**ADDRESS LOW:** These lower order address inputs are used to generate the column address for the internal address multiplexer. In iAPX 286 mode (CFS = 1), these addresses are latched internally.
AL1	4	56	I	
AL2	3	57	I	
AL3	2	58	I	
AL4	1	59	I	
AL5	47	63	I	
AL6	46	64	I	
AL7	45	66	I	
AL8	44	67	I	
AH0	43	2	I	**ADDRESS HIGH:** These higher order address inputs are used to generate the row address for the internal address multiplexer. In iAPX 286 mode, these addresses are latched internally.
AH1	42	3	I	
AH2	41	4	I	
AH3	40	5	I	
AH4	39	6	I	
AH5	38	7	I	
AH6	37	8	I	
AH7	35	12	I	
AH8	34	13	I	
BS	6	50	I	**BANK SELECT:** This input is used to select one of the two banks of the dynamic RAM array.
AO0	7	49	O	**ADDRESS OUTPUTS:** These outputs are designed to provide the row and column addresses, of either the CPU or the refresh counter, to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers. However, they typically need series resistors to match impedances.
AO1	8	48	O	
AO2	9	47	O	
AO3	10	46	O	
AO4	11	45	O	
AO5	13	41	O	
AO6	14	40	O	
AO7	15	39	O	
AO8	16	38	O	
$\overline{RAS0}$	19	33	O	**ROW ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the row address, present on the AO0–8 pins. These outputs are selected by the BS pin. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{RAS1}$	18	36	O	
$\overline{CAS0}$	21	30	O	**COLUMN ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the column address, present on the AO0–8 pins, These outputs are selected by the BS pin. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{CAS1}$	20	31	O	
RESET	23	28	I	**RESET:** This active high signal causes all internal counters to be reset. Upon release of RESET, data appearing at the PDI pin is clocked-in by the PCLK output. The states of the PDI, PCTL, and RFRQ pins are sampled by RESET going inactive and are used to program the 82C08. An 8-cycle dynamic RAM warm-up is performed after clocking PDI bits into the 82C08.
WE/ PCLK	25	24	O	**WRITE ENABLE/PROGRAMMING CLOCK:** Immediately after a RESET this pin becomes PCLK and is used to clock serial programming data into the PDI pin. After the 82C08 is programmed this active high signal provides the dynamic RAM array the write enable input for a write operation.

Table 1. Pin Description (Continued)

Symbol	DIP Pin	PLCC	Type	Name and Function
$\overline{AACK}$/ $\overline{XACK}$	26	23	O	**ADVANCE ACKNOWLEDGE/TRANSFER ACKNOWLEDGE:** When the X programming bit is set to logic 0 this pin is $\overline{AACK}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the S program-bit for synchronous or asynchronous operation. The S programming bit determines whether this strobe will be early or late. If another dynamic RAM cycle is in progress at the time of the new request, the $\overline{AACK}$ is delayed. When the X programming bit is set to logic 1 this pin is $\overline{XACK}$ and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle. $\overline{XACK}$ is a MULTIBUS compatible signal.
PCTL	27	22	I	**PORT CONTROL:** This pin is sampled on the falling edge of RESET. It configures the 82C08 to accept command inputs or processor status inputs. If PCTL is low after RESET the 82C08 is programmed to accept bus/multibus command inputs or iAPX 286 status inputs. If PCTL is high after RESET the 82C08 is programmed to accept status inputs from iAPX 86 or iAPX 186 type processors. The S2 status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 inputs. When programmed to accept bus commands or iAPX 286 status inputs, it should be tied low or it may be connected to INHIBIT when operating with MULTIBUS.
$\overline{PE}$	28	21	I	**PORT ENABLE:** This pin serves to enable a RAM cycle request. It is generally decoded from the address bus.
$\overline{WR}$	29	20	I	**WRITE:** This pin is the write memory request command input. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{RD}$	30	19	I	**READ:** This pin is the read memory request command pin. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
CLK	31	16	I	**CLOCK:** This input provides the basic timing for sequencing the internal logic.
RFRQ	32	15	I	**REFRESH REQUEST:** This input is sampled on the falling edge of RESET. If RFRQ is high at RESET then the 82C08 is programmed for internal-refresh request or external-refresh request with failsafe protection. If RFRQ is low at RESET then the 82C08 is programmed for external-refresh without failsafe protection or burst refresh. Once programmed the RFRQ pin accepts signals to start an external-refresh with failsafe protection or external-refresh without failsafe protection or a burst refresh. RFRQ is also sampled when PDD is activated. When RFRQ = 1 it will cause 3 burst refresh cycles.
PDI	33	14	I	**PROGRAM DATA INPUT:** This input is sampled by RESET going low. It programs the various user selectable options in the 82C08. The PCLK pin shifts programming data into the PDI input from an external shift register. This pin may be strapped low to a default iAPX 186 mode configuration or high to a default iAPX 286 mode configuration.
*PDD	17	37	I	**POWER DOWN DETECT:** This input is sampled before every memory cycle to inform the 82C08 of system detection of power failure. When active, the 82C08 remains in power down mode and performs memory refresh only ($\overline{RAS}$-only refresh). In power down mode the 82C08 uses PDCLK for timing and VPD for power.

Table 1. Pin Description (Continued)

Symbol	DIP Pin	PLCC	Type	Name and Function
*PDCLK	22	29	I	**POWER DOWN CLOCK:** This pin is used as a clock for internal refresh circuits during power down. The input can be asynchronous to pin 31. Extended refresh is achieved by slowing down this clock. This pin should be grounded if not used.
*V_{CC}/V_{PD}	48	61, 62	I	**POWER:** Power supply for internal logic. This should be held active during power down, and normal operation.
V_{CC}	24	26, 27	I	**POWER:** Supply for drivers. Need not be held active during power down.
V_{SS}	12 36	9, 10, 11, 42, 43, 44	I I	GROUND GROUND
NC	—	17, 18, 1, 25, 32, 34, 35, 51, 53, 54, 60, 65, 68		
V_{CCS}		52		Connect to V_{PP}, Pin 48 for PLCC package.

*Different function than the HMOS 8208.

GENERAL DESCRIPTION

The Intel 82C08 Dynamic RAM Controller is a microcomputer peripheral device which provides the necessary signals to address, refresh, and directly drive 64K and 256K dynamic RAMs. It is compatible with static column or ripple mode DRAMs in the normal mode. It does not support the fast transfer mode of these DRAMs.

The 82C08 supports several microprocessor interface options including synchronous and asynchronous operations for iAPX 86, iAPX 186, iAPX 286, and MULTIBUS. The 82C08 will also interface to non-Intel microprocessors.

The 82C08 is a CHMOS version of the 8208 and is pin compatible with it. Three pins—17, 22, and 48—of the 82C08 are different from the 8208. They provide a power down mode that allows the system to run at a much lower ICC. In this mode, the 82C08 refreshes the DRAM using battery backup. The power down current (I_{PD}) that is drawn by the 82C08 is very small compared to the I_{CC} which allows memory to be kept alive with a battery. A separate refresh clock, pin 22, allows the designer to take advantage of RAMs that permit extended memory refresh.

The 82C08 also has some timing changes versus the 8208. In order to eliminate the external bus latches, both WE and $\overline{CAS}$ timings are shortened. These timing changes are backwards-compatible for 8208 designs.

FUNCTIONAL DESCRIPTION

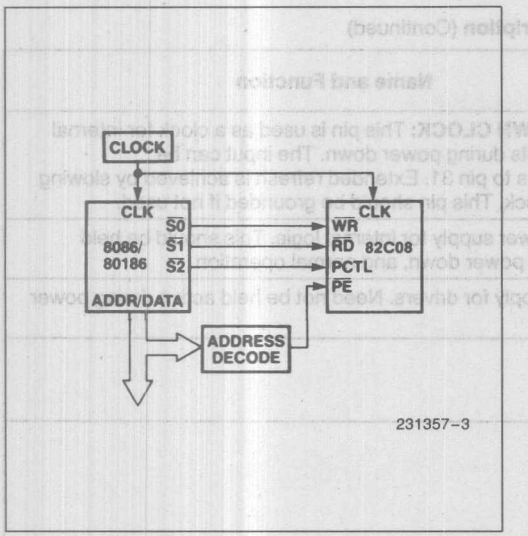
Processor Interface

The 82C08 has control circuitry capable of supporting one of several possible bus structures. The 82C08 may be programmed to run synchronous or asynchronous to the processor clock. The 82C08 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186/188 and iAPX 286. When the 82C08 is programmed to run in asynchronous mode, the 82C08 inserts the necessary synchronization circuitry for the $\overline{RD}$, $\overline{WR}$ inputs.

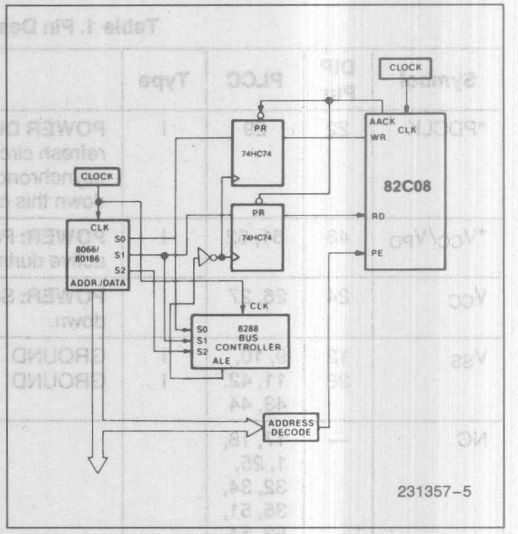
The 82C08 achieves high performance (i.e. no wait states) by decoding the status lines directly from the processor. The 82C08 can also be programmed to receive read or write MULTIBUS commands or commands from a bus controller.

The 82C08 may be programmed to operate synchronously to the processor. It can also be programmed to run at various frequencies. (See Microprocessor Clock Frequency Option.)

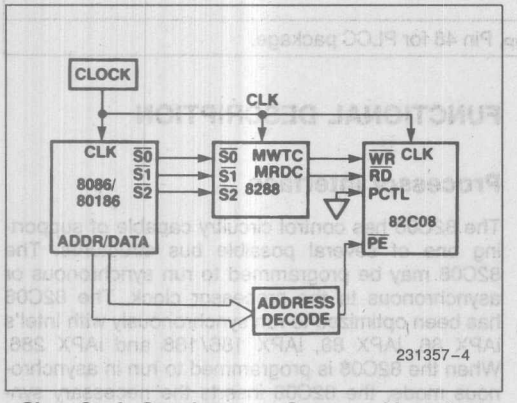
Figure 2 shows the different processor interfaces to the 82C08 using the synchronous or asynchronous mode and status or command interface. Figure 3 shows detailed interfaces to the iAPX 186 and iAPX 286 processors.



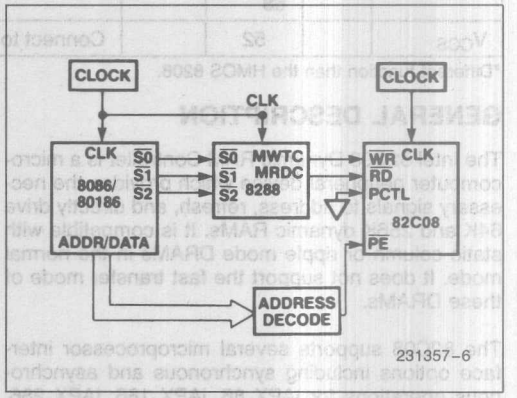
Slow-Cycle Synchronous-Status Interface



Slow-Cycle Asynchronous-Status Interface

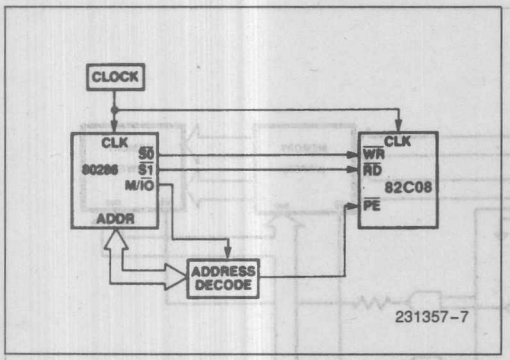


Slow-Cycle Synchronous-Command Interface

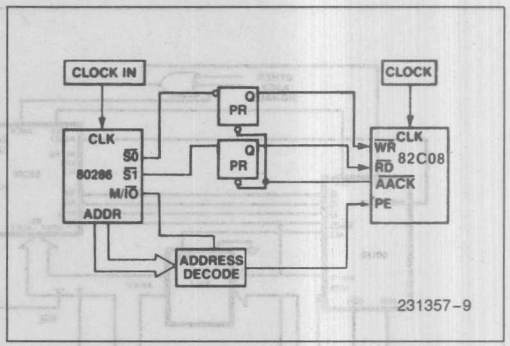


Slow-Cycle Asynchronous-Command Interface

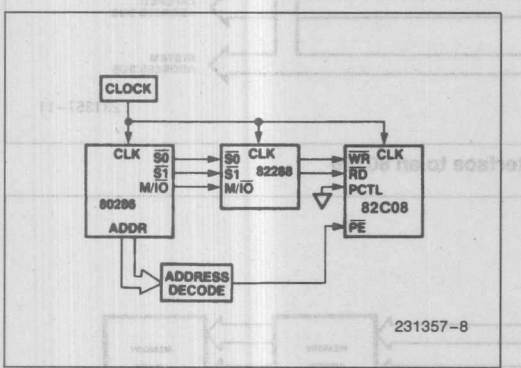
Figure 2A. Slow-cycle (CFS = 0) Port Interfaces Supported by the 82C08



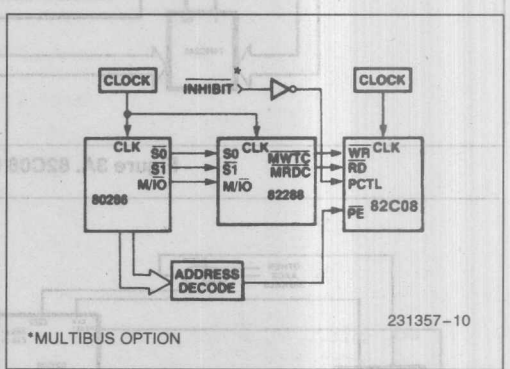
Fast-Cycle Synchronous-Status Interface



Fast-Cycle Asynchronous-Status Interface



Fast-Cycle Synchronous-Command Interface



Fast-Cycle Asynchronous-Command Interface

Figure 2B. Fast-cycle (CFS = 1) Port Interfaces Supported by the 82C08

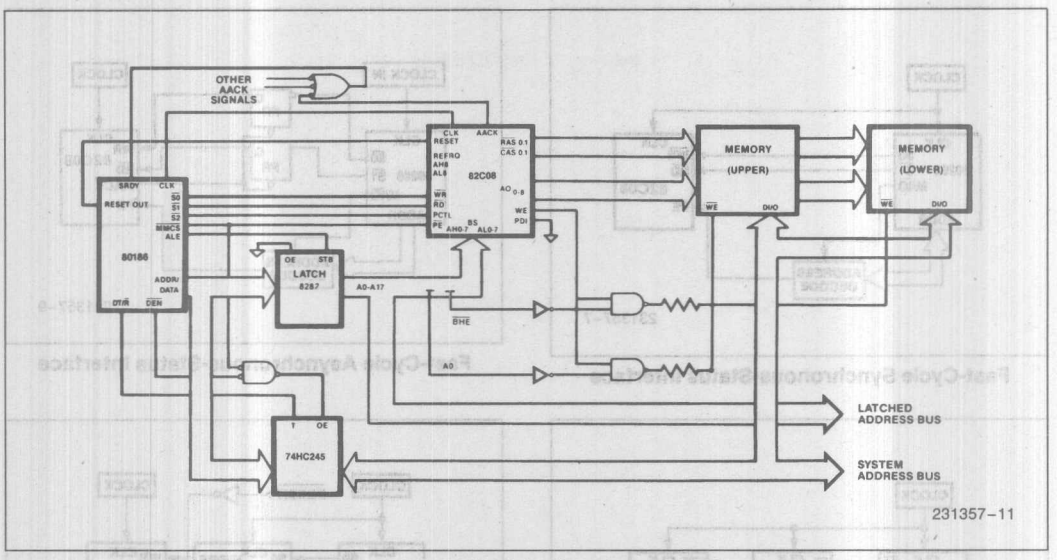


Figure 3A. 82C08 Interface to an 80186

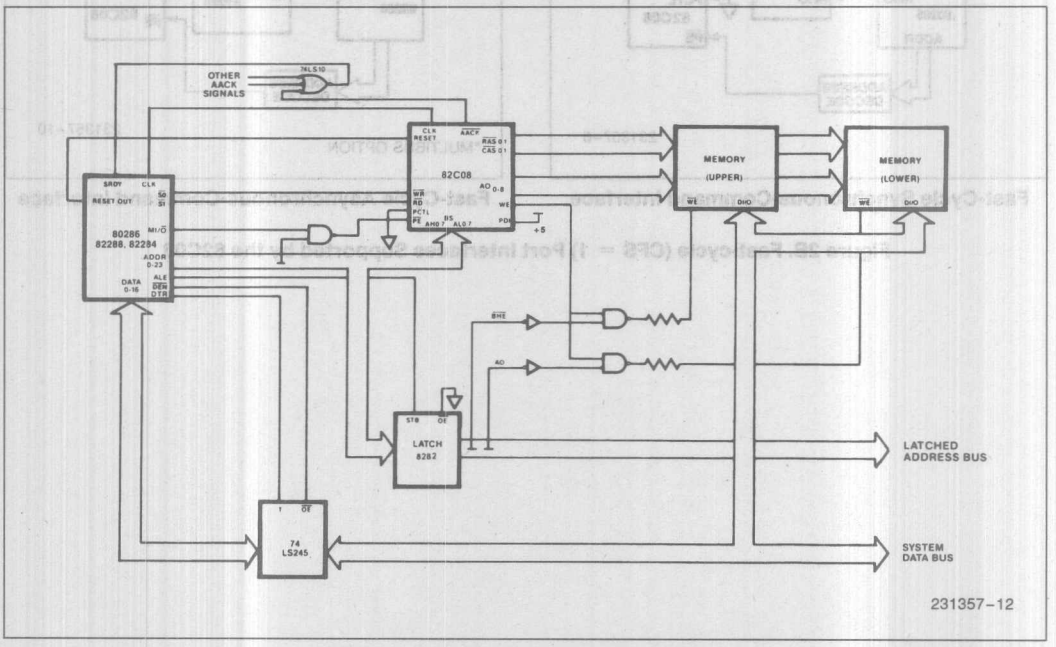
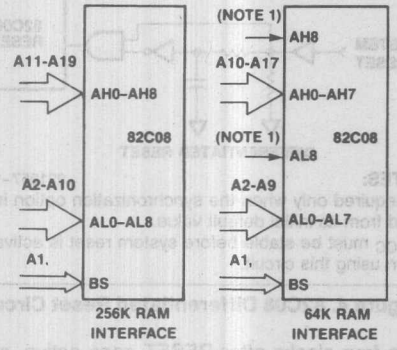


Figure 3B. 82C08 Interface to an 80286

Dynamic RAM Interface

The 82C08 is capable of addressing 64K and 256K dynamic RAMs. Figure 3 shows the connection of the processor address bus to the 82C08 using the different RAMs.



NOTES:
1. Unassigned address input pins should be strapped high.
2. A0 along with BHE are used to select a byte within a processor word.
3. Low order address bit is used as a bank select input so that consecutive memory access requests are to alternate banks allowing bank interleaving of memory cycles.

Figure 3. Processor Address Interface to the 82C08 Using 64K, and 256K RAMS

The 82C08 divides memory into two banks, each bank having its own Row ($\overline{RAS}$) and Column ($\overline{CAS}$) Address Strobe pair. This organization permits RAM cycle interleaving. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM precharge period of the previous cycle. Hiding the precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in the alternate banks.

Successive data access to the same bank cause the 82C08 to wait for the precharge time of the previous RAM cycle. But when the 82C08 is programmed in an iAPX 186 synchronous configuration, consecutive cycles to the same bank do not result in additional wait states (i.e. 0 wait state).

If not all RAM banks are occupied, the 82C08 can be programmed to reassign the $\overline{RAS}$ and $\overline{CAS}$ strobes to allow using wider data words without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers.

Table 2 shows the bank selection decoding and the corresponding $\overline{RAS}$ and $\overline{CAS}$ assignments. For example, if only one RAM bank is occupied, then the two $\overline{RAS}$ and $\overline{CAS}$ strobes are activated with the same timing.

Table 2. Bank Selection Decoding and Word Expansion

Program Bit RB	Bank Input BS	82C08 $\overline{RAS}/\overline{CAS}$ Pair Allocation
0	0	$\overline{RAS}_{0,1}$, $\overline{CAS}_{0,1}$ to Bank 0
0	1	Illegal
1	0	$\overline{RAS}_0$, $\overline{CAS}_0$ to Bank 0
1	1	$\overline{RAS}_1$, $\overline{CAS}_1$ to Bank 1

Program bit RB is not used to check the bank select input BS. The system design must protect from accesses to "illegal", non-existent banks of memory by deactivating the PE input when addressing an "illegal", non-existent bank of memory.

The 82C08 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option.)

Memory Initialization

After programming, the 82C08 performs eight RAM "wake-up" cycles to prepare the dynamic RAM for proper device operation.

Refresh

The 82C08 provides an internal refresh interval counter and a refresh address counter to allow the 82C08 to refresh memory. The 82C08 has a 9-bit internal refresh address counter which will refresh 128 rows every 2 milliseconds, 256 rows every 4 milliseconds or 512 rows every 8 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 82C08 may be programmed for any of five different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst refresh modes, or no refresh. (See Refresh Options.)

It is possible to decrease the refresh time interval by 10%, 20% or 30%. This option allows the 82C08 to compensate for reduced clock frequencies. Note

that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options.)

External Refresh Requests after RESET

External refresh requests are not recognized by the 82C08 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper dynamic RAM operation. The time it takes for the 82C08 to recognize a request is shown below.

eg. 82C08 System Response:

TRESP = TPROG + TPREP

where: TPROG = (40) (TCLCL) programming time

TPREP = (8) (32) (TCLCL) RAM warm-up time

if TCLCL = 125 ns then TRESP = 37 μs

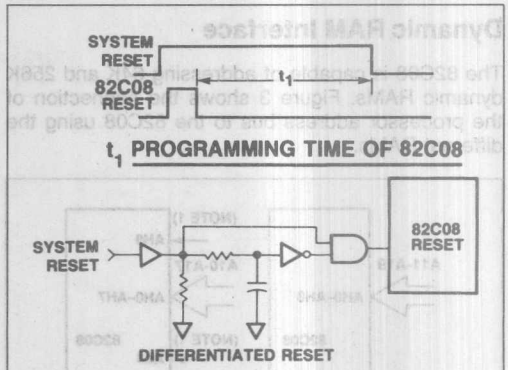
Reset

RESET is an asynchronous input, its falling edge is used by the 82C08 to directly sample the logic levels of the PCTL, RFRQ, and PDI inputs. The internally synchronized falling edge of reset is used to begin programming operations (shifting in the contents of the external shift register, if needed, into the PDI input).

Differentiated reset is unnecessary when the default synchronization programming is used.

Until programming is complete the 82C08 latches but does not respond to command or status inputs. A problem may occur if the S bit is programmed inconsistently from the Command which was latched before programming was completed. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 82C08.

The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 82C08. The differentiated reset pulse first resets the 82C08, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 82C08 completes its programming. Figure 4 illustrates a circuit to accomplish this task.



NOTES:
1. Required only when the synchronization option is altered from its initial default value.
2. V_{CC} must be stable before system reset is activated when using this circuit.

Figure 4. 82C08 Differentiated Reset Circuit

Within four clocks after RESET goes active, all the 82C08 outputs will go high, except for AO0–2, which will go low.

OPERATIONAL DESCRIPTION

Programming the 82C08

The 82C08 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTL, RFRQ, and PDI pins.

Status/Command Mode

The processor port of the 82C08 is configured by the states of the PCTL pin. Which interface is selected depends on the state of the PCTL pin at the end of reset. If PCTL is high at the end of reset, the 8086/80186 Status interface is selected; if it is low, then the MULTIBUS or Command interface is selected.

The status lines of the 80286 are similar in code and timing to the Multibus command lines, while the status code and timing of the 8086 and 8088 are identical to those of the 80186 and 80188 (ignoring the differences in clock duty cycle). Thus there exists two interface configurations, one for the 80286 status or Multibus memory commands, which is called the Command interface, and one for 8086,

8088, 80186 or 80188 status, called the 8086 Status interface. The Command interface can also directly interface to the command lines of the bus controllers for the 8086, 8088, 80186 and the 80286.

The 80186 Status interface allows direct decoding of the status lines for the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded.

Table 3A. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286*
0	0	0	INTERRUPT	INTERRUPT
0	0	1	I/O READ	I/O READ
0	1	0	I/O WRITE	I/O WRITE
0	1	1	HALT	IDLE
1	0	0	INSTRUCTION FETCH	HALT
1	0	1	MEMORY READ	MEMORY READ
1	1	0	MEMORY WRITE	MEMORY WRITE
1	1	1	IDLE	IDLE

* Refer to 80286 pin description table

Table 3B. 82C08 Response

82C08 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086/80186 Status Interface	80286 Status or Command Interface
0	0	0	IGNORE	IGNORE*
0	0	1	IGNORE	READ
0	1	0	IGNORE	WRITE
0	1	1	IGNORE	IGNORE
1	0	0	READ	IGNORE
1	0	1	READ	INHIBIT
1	1	0	WRITE	INHIBIT
1	1	1	IGNORE	IGNORE

*Illegal with CFS = 0

Refresh Options

Immediately after system reset, the state of the RFRQ input pin is examined. If RFRQ is high, the 82C08 provides the user with the choice between self-refresh and user-generated refresh with failsafe protection. Failsafe protection guarantees that if the user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the RFRQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 82C08 to generate internal refresh requests, it is necessary only to strap the RFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the RFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 82C08. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold RFRQ low until after reset. Thereafter, bringing RFRQ high for one clock period will cause a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., RFRQ is kept low until after reset). Thereafter, bringing RFRQ high for at least two clock periods will cause a burst of up to 128 row address locations to be refreshed. A refresh request is not recognized until a previous request has been serviced (i.e. burst is completed).

No Refresh

It is necessary to hold RFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping RFRQ low.

Option Program Data Word

PROGRAMMING FOR SLOW CYCLE

The program data word consists of 9 program data bits, PD0–PD8. If the first program data bit, PD0 is

set to logic 0, the 82C08 is configured to support iAPX 186, 188, 86, or 88 systems. The remaining bits, PD1–PD8, may then be programmed to optimize a selected system configuration. A default of all zeros in the remaining program bits optimizes the 82C08 timing for 8 MHz Intel CPUs using 150 ns (or faster) dynamic RAMs with no performance penalty.

PROGRAMMING FOR FAST CYCLE

If the first program data bit is set to logic 1, the 82C08 is configured to support iAPX 286 systems (Command mode). A default of all ones in the program bits optimizes the 82C08 timing for an 8 MHz 286 using 120 ns DRAMs at zero wait states. Note that the programming bits PD1–8 change polarity according to PD0. This ensures the same choice of options for both default modes.

Table 4A shows the various options that can be programmed into the 82C08.

Table 4A. Program Data Word

Program Data Bit	Name		Polarity/Function
	PD0 = 0	PD0 = 1	
PD0	CFS	CFS	CFS = 0 SLOW CYCLE CFS = 1 FAST CYCLE
PD1	$\overline{S}$	S	$\overline{S}$ = 0 SYNCHRONOUS* $\overline{S}$ = 1 ASYNCHRONOUS
PD2	$\overline{RFS}$	RFS	$\overline{RFS}$ = 0 FAST RAM* $\overline{RFS}$ = 1 SLOW RAM
PD3	$\overline{RB}$	RB	RAM BANK OCCUPANCY SEE TABLE 2
PD4	CI1	$\overline{CI1}$	COUNT INTERVAL BIT 1; SEE TABLE 6
PD5	CI0	$\overline{CI0}$	COUNT INTERVAL BIT 0; SEE TABLE 6
PD6	$\overline{PLS}$	PLS	$\overline{PLS}$ = 0 LONG REFRESH PERIOD* $\overline{PLS}$ = 1 SHORT REFRESH PERIOD
PD7	$\overline{FFS}$	FFS	$\overline{FFS}$ = 0 FAST CPU FREQUENCY* $\overline{FFS}$ = 1 SLOW CPU FREQUENCY
PD8	X	$\overline{X}$	X = 0 $\overline{AACK}$* X = 1 $\overline{XACK}$

* Default in both modes

Using an External Shift Register

The 82C08 may be programmed by using an external shift register with asynchronous load capability such as a 74HC165. The reset pulse serves to parallel load the shift register and the 82C08 supplies the clocking signal (PCLK) to shift the data into the PDI programming pin. Figure 6 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 82C08 via the PDI pin (33), and clock is provided by the WE/PCLK pin (25), which generates a total of 9 clock pulses.

WE/PCLK is a dual function pin. During programming, it serves to clock the external shift register, and after programming is completed, it reverts to the write enable RAM control output pin. As the pin changes state to provide the write enable signal to the dynamic RAM array, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 7 illustrates the timing requirements of the shift register.

Default Programming Options

After reset, the 82C08 serially shifts in a program data word via the PDI pin. This pin may be strapped low or high, or connected to an external shift register. Strapping PDI low causes the 82C08 to default to the iAPX 186 system configuration, while high causes a default to the iAPX 286 configuration. Table 4B shows the characteristics of the default configuration for Fast Cycle (PDI = 1) and Slow Cycle (PDI = 0). If further system flexibility is needed, one external shift register, like a 74HC165, can be used to tailor the 82C08 to its operating environment.

Table 4B. Default Programming

Synchronous interface
Fast RAM (Note 1)
2 RAM banks occupied
128 row refresh in 2 ms; 256 in 4 ms, 512 in 8 ms
Fast processor clock frequency
Advanced ACK strobe

NOTE:
1. For iAPX 86/186 systems either slow or fast (150 or 100 ns) RAMS will run at 8 MHz with zero wait states.

Synchronous/Asynchronous Mode (S program bit)

The 82C08 may be configured to accept synchronous or asynchronous commands ($\overline{RD}$, $\overline{WR}$, PCTL) and Port Enable ($\overline{PE}$) via the S program bit. The state of the S programming bit determines whether the interface is synchronous or asynchronous.

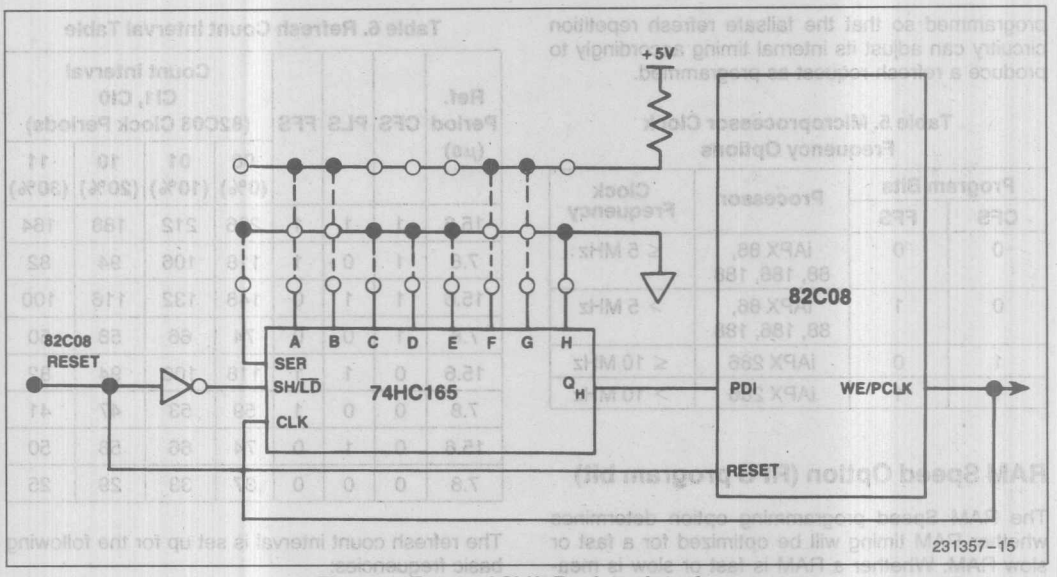
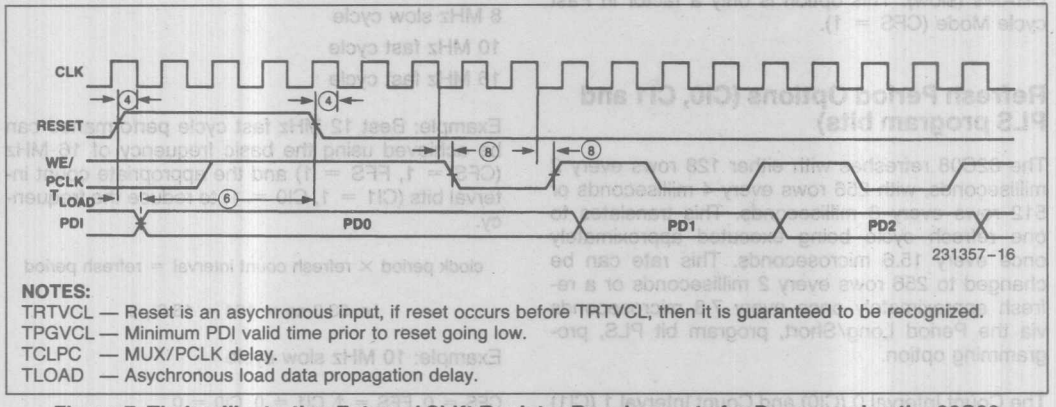


Figure 6. External Shift Register Interface



NOTES:
TRTVCL — Reset is an asychronous input, if reset occurs before TRTVCL, then it is guaranteed to be recognized.
TPGVCL — Minimum PDI valid time prior to reset going low.
TCLPC — MUX/PCLK delay.
TLOAD — Asychronous load data propagation delay.

Figure 7. Timing Illustrating External Shift Register Requirements for Programming the 82C08

While the 82C08 may be configured with either the Status or Command (MULTIBUS) interface in the Synchronous mode, certain restrictions exist in the Asynchronous mode. An Asynchronous-Command interface is directly supported. An Asynchronous-80186/80286 Status interface using the status lines of the 80186/80286 is supported with the use of TTL gates as illustrated in Figure 2. In the 80186 case, the TTL gates are needed to guarantee that status does not appear at the 82C08's inputs too much before address, so that a cycle would start before address was valid. In the case of the 80286, the TTL gates are used for lengthening the Status pulse, as required by the TRWL timing.

Microprocessor Clock Cycle Option (CFS and FFS program bits)

The 82C08 is programmed to interface with microprocessors with "slow cycle" timing like the 8086, 8088, 80186, and 80188, and with "fast cycle" microprocessors like the 80286. The CFS bit is used to select the appropriate timing.

The FFS option is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed. The external clock frequency must be

programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

Table 5. Microprocessor Clock Frequency Options

Program Bits		Processor	Clock Frequency
CFS	FFS		
0	0	iAPX 86, 88, 186, 188	≤ 5 MHz
0	1	iAPX 86, 88, 186, 188	> 5 MHz
1	0	iAPX 286	≤ 10 MHz
1	1	iAPX 286	> 10 MHz

RAM Speed Option (RFS program bit)

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM. Whether a RAM is fast or slow is measured relative to 100 ns DRAMs (fast) or 150 ns DRAMs (slow). This option is only a factor in Fast cycle Mode (CFS = 1).

Refresh Period Options (CI0, CI1 and PLS program bits)

The 82C08 refreshes with either 128 rows every 2 milliseconds, with 256 rows every 4 milliseconds or 512 rows every 8 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option.

The Count Interval 0 (CI0) and Count Interval 1 (CI1) programming options allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the 15.6 or 7.8 microsecond period when the 82C08 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests.

Table 6. Refresh Count Interval Table

Ref. Period (μs)	CFS	PLS	FFS	Count Interval CI1, CI0 (82C08 Clock Periods)			
				00 (0%)	01 (10%)	10 (20%)	11 (30%)
15.6	1	1	1	236	212	188	164
7.8	1	0	1	118	106	94	82
15.6	1	1	0	148	132	116	100
7.8	1	0	0	74	66	58	50
15.6	0	1	1	118	106	94	82
7.8	0	0	1	59	53	47	41
15.6	0	1	0	74	66	58	50
7.8	0	0	0	37	33	29	25

The refresh count interval is set up for the following basic frequencies:

5 MHz slow cycle

8 MHz slow cycle

10 MHz fast cycle

16 MHz fast cycle

Example: Best 12 MHz fast cycle performance can be achieved using the basic frequency of 16 MHz (CFS = 1, FFS = 1) and the appropriate count interval bits (CI1 = 1, CI0 = 1) to reduce the frequency.

clock period × refresh count interval = refresh period

i.e. 83.3 ns × 164 = 13.6 μs

Example: 10 MHz slow cycle

CFS = 0, FFS = 1, CI1 = 0, CI0 = 0

i.e. 100 ns × 118 = 11.8 μs

Processor Timing

In order to run without wait states, $\overline{AACK}$ must be used and connected to the $\overline{SRDY}$ input of the appropriate bus controller. $\overline{AACK}$ is issued relative to a point within the RAM cycle and has no fixed relationship to the processors's request. The timing is such, however, that the processor will run without wait states, barring refresh cycles. In slow cycle, fast RAM configurations (8086, 80186), $\overline{AACK}$ is issued on the same clock cycle that issues $\overline{RAS}$.

Port Enable ($\overline{PE}$) set-up time requirements depend on whether the 82C08 is configured for synchronous

or asynchronous, fast or slow cycle operation. In a synchronous fast cycle configuration, $\overline{PE}$ is required to be set-up to the same clock edge as the commands. If $\overline{PE}$ is true (low), a RAM cycle is started; if not, the cycle is not started until the $\overline{RD}$ or $\overline{WR}$ line goes inactive and active again.

In asychronous operation, $\overline{PE}$ is required to be set-up to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command) -to-$\overline{PE}$ delay time, thus allowing for more external decode time than is available in synchronous operation.

The minimum synchronization delay is the additional amount that $\overline{PE}$ must be held valid. If $\overline{PE}$ is not held valid for the maximum synchronization delay time, it is possible that $\overline{PE}$ will go invalid prior to the status or command being synchronized. In such a case the 82C08 may not start a memory cycle. If a memory cycle intended for the 82C08 is not started, then no acknowledge ($\overline{AACK}$ or $\overline{XACK}$) is issued and the processor locks up in endless wait states.

Memory Acknowledge ($\overline{AACK}$, $\overline{XACK}$)

Two types of memory acknowledge signals are supplied by the 82C08. They are the Advanced Acknowledge strobe ($\overline{AACK}$) and the Transfer Acknowledge strobe ($\overline{XACK}$). The S programming bit optimizes $\overline{AACK}$ for synchronous operation ("early" $\overline{AACK}$) or asynchronous operation ("late" $\overline{AACK}$). Both the early and late $\overline{AACK}$ strobes are two clocks long for CFS = 0 and three clocks long for CFS = 1.

The $\overline{XACK}$ strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the MULTIBUS requirements. $\overline{XACK}$ is removed asynchronously by the command going inactive.

Since in an asynchronous operation the 82C08 removes read data before late $\overline{AACK}$ or $\overline{XACK}$ is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation data latching is unnecessary, since the 82C08 will not remove data until the CPU has read it.

If the X programming bit is high, the strobe is configured as $\overline{XACK}$, while if the bit is low, the strobe is configured as $\overline{AACK}$.

Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Thus, the advanced acknowledge may also serve as a RAM cycle timing indicator.

General System Considerations

1. The $\overline{RAS}$0, 1, $\overline{CAS}$0, 1, and AO0–8 output buffers are designed to directly drive the heavy capacitive loads of the dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment it is necessary to match the output impedance with the RAM array by using series resistors. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application.
2. Although the 82C08 has programmable options, in practice there are only a few choices the designer must make. For iAPX 86/186 systems (CFS = 0) the C2 default mode (pin 33 tied low) is the best choice. This permits zero wait states at 8 and 10 MHz with 150 ns DRAMs. The only consideration is the refresh rate, which must be programmed if the CPU is run at less than 8 MHz.

   For iAPX 286 systems (CFS = 1) the designer must choose between configuration C0 ($\overline{RFS}$ = 0) and C1 ($\overline{RFS}$ = 1, $\overline{FFS}$ = 0). C0 permits zero wait state, 8 MHz iAPX 286 operation with 120 ns DRAMs. However, for consecutive reads, this performance depends on interleaving between two banks. The C1 configuration trades off 1 wait state performance for the ability to use 150 ns DRAMs. 150 ns DRAMs can be supported by the C0 configuration using 7 MHz iAPX 286.
3. For non-Intel microprocessors, the asynchronous command mode would be the best choice, since Intel status lines are not available. To minimize the synchronization delay, the 82C08 should use a 16 MHz clock. The preferred timing configuration is C0.

Table 7. Memory Acknowledge Summary

	Synchronous	Asynchronous	XACK
Fast Cycle	AACK Optimized for Local 80286 (early)	AACK Optimized for Remote 80286 (late)	Multibus Compatible
Slow Cycle	AACK Optimized for Local 8086/186 (early)	AACK Optimized for Remote 8086/186 (late)	Multibus Compatible

POWER DOWN

During Power Down (PD) mode, the 82C08 will perform refresh cycles to preserve the memory content. Two pins are dedicated to this feature, PDD (Power Down Detect) and PDCLK (Power Down Clock). PDD is used to inform the 82C08 of a system power failure, and will remain active as long as the power is down. It is the system's responsibility to detect power failure and to supply this signal. PDCLK is used to supply the clock during power down for the 82C08 refresh circuits. It is the system's responsibility to supply this clock.

Power Supplies

Power down is achieved by eliminating the clock from all the 82C08 circuits that are not participating in the refresh generation. The 82C08 has two power pins (V_{CC}'s), one supplies power to the output buffers and the other, to 82C08 logic. All the active circuits during power down are connected to the logic V_{CC}, including the active output buffers. Therefore, it is the user's choice to connect only the logic V_{CC} pin to the back-up power supply, or to connect both pins to it. It is recommended, however, to connect both pins to the same power supply in order to simplify and to shorten the power up time.

Extended Refresh at Power Down (PD)

To reduce power dissipation during PD, 82C08 will support the extended refresh cycle of the Intel 51CXXL (e.g. 51C64L). In this mode, the refresh period can be extended up to 64 milliseconds versus 4 milliseconds in non-extended cycles. This is achieved by slowing down the PDCLK frequency.

The user should take into consideration that when supporting extended refresh during PD, the dynamic RAM must be refreshed completely within 4 milliseconds, without active cycles, both before going into and after coming out of extended refresh. The 82C08 has the option of performing burst refresh of all the memory whenever the user cannot guarantee the 4 milliseconds idle interval. This is achieved by performing 3 consecutive burst refresh cycles, activated internally by the 82C08.

The option of refreshing all the memory is enabled in failsafe mode configuration (RFRQ input high at reset). When 82C08 detects power down, (high level at PDD) it examines the RFRQ input. High level at the RFRQ input will cause 3 PD burst refresh cycles to be performed. The user should supply the power and the system clock during the time interval of the 3 PD burst cycles, e.g. 4700 (fast cycle) or 3100 (slow cycle) clock cycles after activating PDD. Low level at RFRQ input enables the 82C08 to enter power down immediately without executing any bursts.

Power Down Procedure

The 82C08 will preserve the memory content during the entire period of the system operation. Upon detection of power down, the 82C08 will save internally its configuration status and the refresh address counter content, execute 3 burst refresh cycles. (If it is programmed to failsafe mode and the RFRQ input level is high), it will switch the internal clock from the system clock (CLK) to the power down clock (PDCLK) and will continue the refresh to the next address location. (See Figure 11.)

When power is up again (PDD input deactivated), the 82C08 will issue internal reset which will not reprogram the device and will not clear the refresh address counter, and therefore, refresh will continue to the next address location. After the internal reset, 82C08 performs 3 PD burst refresh cycles which refresh the whole memory, as at entering extended PD. This is done to give the 82C08 enough time to wake up. Notice, at the time interval of 4700 (fast cycle) or 3100 (slow cycle) clocks after power recovering no memory access will be performed.

82C08 Outputs on Power Down

Four of the 82C08 outputs are not activated during power down, $\overline{AACK}$, $\overline{CAS}0-1$ and WE. All these outputs will be forced to a non-active state, $\overline{AACK}$ and $\overline{CAS}0-1$ will be forced high and WE will be forced low (External NAND buffer is used to drive the $\overline{WE}$ DRAM inputs, hence a high level on the DRAM inputs). The other 82C08 outputs, AO0–9 and $\overline{RAS}0-1$, will switch to perform the memory refresh in a "$\overline{RAS}$-ONLY REFRESH CYCLE." The $\overline{RAS}$ outputs internal pull-ups assure high levels on these outputs, as close as possible to V_{CC}, for low DRAM power. The size of the output buffers, in power down, is smaller than the normal size, and therefore, the speed of these buffers is slower. It is done in order to reduce the speed of charging and discharging the outputs and hence reduce spikes on the power lines. It is required especially in power down, since there is only one power supply pin active which drives the output buffers as well as the internal logic.

All the device inputs, beside PDD, PDCLK, and RESET will be ignored during power down.

During power down burst refresh the 82C08 performs up to 256 refreshes. Whereas during standard burst refresh the 82C08 performs up to 128 refreshes. The power down burst refresh feature allows the 82C08 to support extended refreshes of some DRAMs, configured as 512 rows.

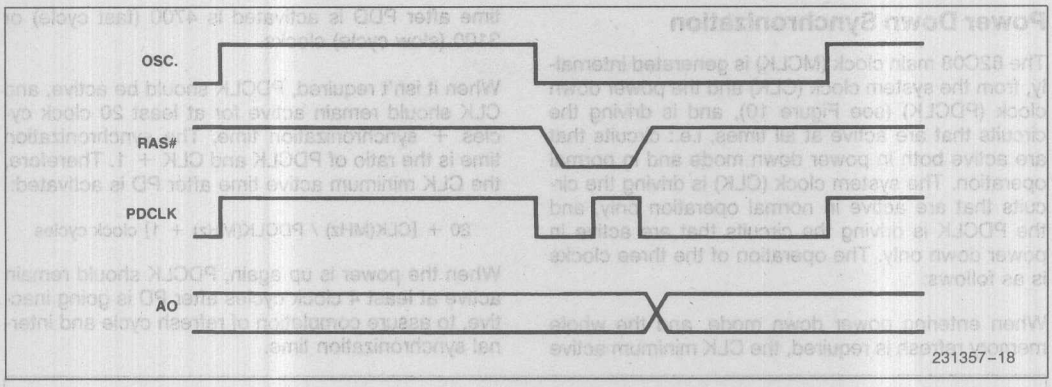


Figure 8

Power Down Detect

As previously mentioned, the PDD input will be supplied by the system to inform the 82C08 of a power failure. It can be asynchronous since the 82C08 synchronizes it internally. The PDD input will be sampled by the 82C08 before the beginning of every memory cycle but only after the termination of programming and initialization period. The user should guarantee V_{CC} and CLK stable during the programming and initialization period (300 clocks after RESET). If the whole memory refresh is required (for extended refresh) then V_{CC} and system clock should be available 4700 (fast cycle) or 3100 (slow cycle) clocks after activating PDD. If it isn't required then 82C08 should wait for present memory cycle completion and synchronization time which will take about 25 system clock cycles.

With PDD going inactive, the 82C08 synchronizes the clock back to the CLK clock, issuing internal reset and will perform 3 PD burst refresh cycles.

NOTE:
The power supplies and the CLK should go up before the PDD is deactivated. All CPU requests will be ignored when PDD is active.

Refresh during Power Down

The 82C08 has two clock pins, CLK is the system clock and PDCLK is the power down clock. PDCLK should be an independent clock which has its own crystal oscillator. When entering power down, the 82C08 will disable the system clock internally and will run with the PDCLK. The system clock will be enabled and the PDCLK will be disabled when power is up. The CLK and PDCLK will be switched internally for the refresh circuits.

During power down, '$\overline{\text{RAS}}$-ONLY REFRESH' will be performed by the 82C08. The time interval between refreshes is 5 PDCLKs and this is fixed for all applications. However, the 82C08 can support the extended refresh (up to 64 ms) by slowing down the PDCLK frequency.

During the power down refresh cycle, $\overline{\text{RAS}}$ will be activated for one PDCLK cycle only. In extended refresh, the PDCLK frequency will be below 50 kHz and this will cause a long duration of the $\overline{\text{RAS}}$ signal which will increase the DRAM's current rapidly. To minimize the $\overline{\text{RAS}}$ low pulse, the two RC networks shown in Figure 9 are designed to insert one very fast (1 μs) cycle whenever $\overline{\text{RAS}}$ is low (see Figure 8). The time constant of RC1 and RC2 should be centered around 300 ns and 100 ns respectively.

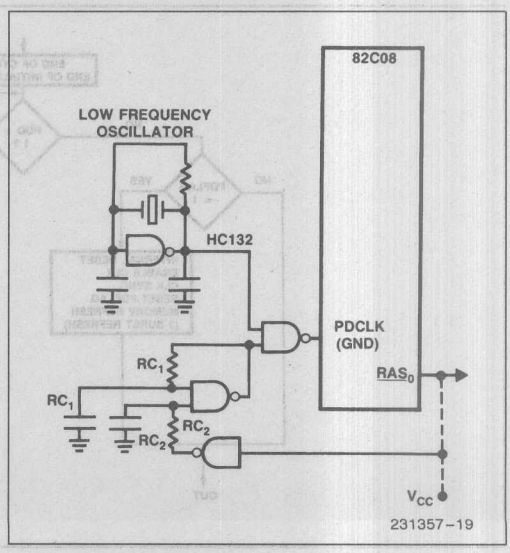


Figure 9. Low Frequency Oscillator

Power Down Synchronization

The 82C08 main clock (MCLK) is generated internally, from the system clock (CLK) and the power down clock (PDCLK) (see Figure 10), and is driving the circuits that are active at all times, i.e.: circuits that are active both in power down mode and in normal operation. The system clock (CLK) is driving the circuits that are active in normal operation only, and the PDCLK is driving the circuits that are active in power down only. The operation of the three clocks is as follows:

When entering power down mode, and the whole memory refresh is required, the CLK minimum active time after PDD is activated is 4700 (fast cycle) or 3100 (slow cycle) clocks.

When it isn't required, PDCLK should be active, and CLK should remain active for at least 20 clock cycles + synchronization time. The synchronization time is the ratio of PDCLK and CLK + 1. Therefore, the CLK minimum active time after PD is activated:

20 + [CLK(MHz) / PDCLK(MHz) + 1] clock cycles

When the power is up again, PDCLK should remain active at least 4 clock cycles after PD is going inactive, to assure completion of refresh cycle and internal synchronization time.

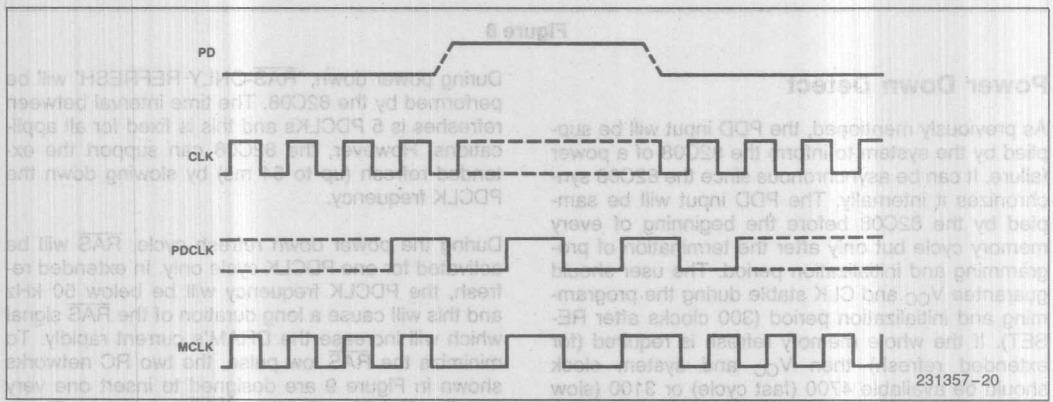


Figure 10

POWER DOWN FLOW

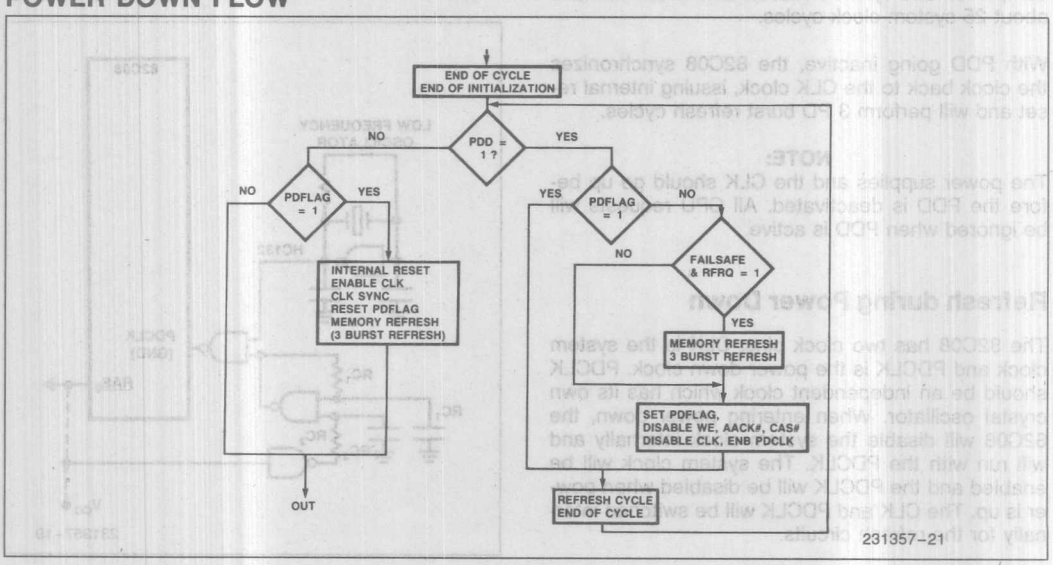


Figure 11

Differences Between 8208 and 82C08

The differences between the HMOS 8208 and the CHMOS 82C08 represent forward compatible enhancements. The 82C08 can be plugged into an 8208 socket without changes.

LOGICAL DIFFERENCES

1. 82C08 has one new feature:

   Power Down (PD)

2. 82C08 supports CMOS DRAMs with T_{RAC} 100, 150

3. Address Mapping:

Outputs	9 Most Significant Bits	9 Least Significant Bits
8208	column address	row address
82C08	row address	column address

4. Slow cycle shortening:

   1). The write cycle is two clocks shorter so consecutive writes will be executed without wait states.

   2) The WE output is two clocks shorter. Therefore, an external latch on the WE output is not necessary.

   3) CAS output is shorter by one clock on the read cycle. This reduces one level of buffers for address/data bus needed in 8208 designs. Read access margins are improved to support non-Intel spec. RAMs.

   4) The address outputs switch from row to column address one clock cycle later in the 82C08 as compared to 8208.

5. Fast cycle shortening:

   1) The write cycle in C0 configuration is shortened by one clock.

   2) For both C0 and C1 synchronous configuration, the CAS signal is shorter by one clock and the activation of RAS is tied to the Φ2 cycle of the 80286. This prevents contention on the data bus.

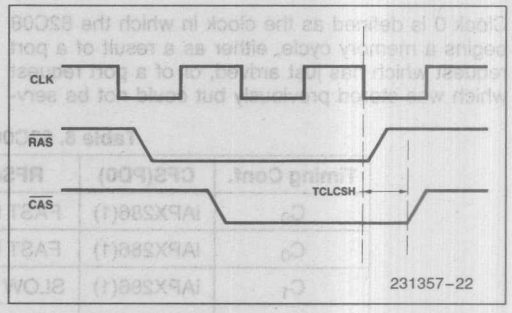
6. Supports Static Column or Ripplemode DRAMs.

ELECTRICAL DIFFERENCES

1. AC parameters:

   1) $\overline{CAS}$ delay: In C2 synchronous read cycle, the $\overline{CAS}$ is deactivated by some delay from clock falling edge (TCLCSH timing) as in the following diagram:

   In C2 write cycles the $\overline{CAS}$ activation is triggered by the clock falling edge with a delay of 35 ns from the clock. For 8208 the delay is TP/1.8 + 53.



   2) 82C08 has an additional timing parameter TARH column address to $\overline{RAS}$ ↑ hold time.

2. DC parameters: The difference is in the current consumption.

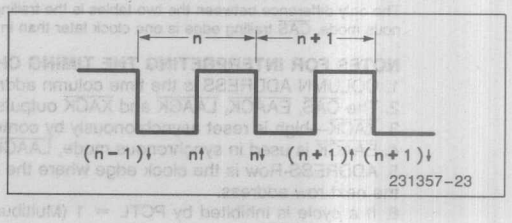
	8208	82C08
I_{CC}	300 mA	30 mA (typical) [10 + 2f] mA (max)
IPD	—	1 mA (max)
I_{SB}	—	2 mA (max)

Configuration Charts

The 82C08 operates in three basic configurations—C0, C1, C2—depending upon the programming of CFS (PD0), $\overline{RFS}$ (PD2), and $\overline{FFS}$ (PD7). Table 8 shows these configurations. These modes determine the clock edges for the 82C08's programmable signals, as shown in Table 9. Finally, Table 10 gives the programmable AC parameters of the 82C08 as a function of configuration. The non-programmable parameters are listed under AC Characteristics.

Using the Timing Charts

The notation used to indicate which clock edge triggers an output transition is "n↑" or "n↓", where "n" is the number of clock periods that have passed since clock 0, the reference clock, and "↑" refers to rising edge and "↓" to falling edge. A clock period is defined as the interval from a clock falling edge to the following falling edge. Clock edges are defined as shown below.



The clock edges which trigger transitions on each 82C08 output are tabulated in Table 9. "H" refers to the high-going transition, and "L" to low-going transition.

Clock 0 is defined as the clock in which the 82C08 begins a memory cycle, either as a result of a port request which has just arrived, or of a port request which was stored previously but could not be serviced at the time of its arrival because the 82C08 was performing another memory cycle. Clock 0 is identified externally by the leading edge of RAS, which is always triggered on 0 ↓.

Table 8. 82C08 Configurations

Timing Conf.	CFS(PD0)	RFS(PD2)	FFS(PD7)	Wait States*
C_0	iAPX286(1)	FAST RAM(1)	20 MHz(1)	0
C_0	iAPX286(1)	FAST RAM(1)	16 MHz(1)	0
C_1	iAPX286(1)	SLOW RAM(0)	16 MHz(1)	1
C_0	iAPX286(1)	FAST RAM(1)	10 MHz (0)	0
C_0	iAPX286(1)	SLOW RAM(0)	10 MHz (0)	0
C_2	iAPX186(0)	DON'T CARE	DON'T CARE	0

* Using $\overline{EAACK}$ (synchronous mode)

Table 9a. Timing Chart — Synchronous Mode

		$\overline{RAS}$		ADDRESS		$\overline{CAS}$		WE		$\overline{EAACK}$	
Cn	Cycle	L	H	Col	Row*	L	H	H	L	L	H
0	RD,RF	0↓	3↓	0↓	3↓	1↓	3↓			1↓	4↓
	WR	0↓	4↓	0↓	3↓	2↓	4↓	1↓	4↓	1↓	4↓
1	RD,RF	0↓	4↓	0↓	4↓	1↓	5↓			2↓	5↓
	WR	0↓	5↓	0↓	4↓	2↓	5↓	1↓	5↓	2↓	5↓
2	RD,RF	0↓	2↓	0↓	3↓	0↓	2↓			0↓	2↓
	WR	0↓	2↓	0↓	3↓	1↓	3↓	0↓	2↓	0↓	2↓

Table 9b. Timing Chart — Asynchronous Mode

		$\overline{RAS}$		ADDRESS		$\overline{CAS}$		WE		$\overline{LAACK}$		$\overline{XAACK}$	
Cn	Cycle	L	H	Col	Row*	L	H	H	L	L	H	L	H
0	RD,RF	0↓	3↓	0↓	3↓	1↓	4↓			2↓	5↓	3↓	$\overline{RD}$
	WR	0↓	4↓	0↓	3↓	2↓	4↓	1↓	4↓	1↓	4↓	3↓	$\overline{WR}$
1	RD,RF	0↓	4↓	0↓	4↓	1↓	6↓			2↓	5↓	4↓	$\overline{RD}$
	WR	0↓	5↓	0↓	4↓	2↓	5↓	1↓	5↓	1↓	4↓	3↓	$\overline{WR}$
2	RD,RF	0↓	2↓	0↓	3↓	0↓	3↓			1↓	3↓	2↓	$\overline{RD}$
	WR	0↓	2↓	0↓	3↓	1↓	3↓	0↓	2↓	1↑	3↑	2↓	$\overline{WR}$

The only difference between the two tables is the trailing edge of $\overline{CAS}$ for all read cycle configurations. In asynchronous mode, $\overline{CAS}$ trailing edge is one clock later than in synchronous mode.

NOTES FOR INTERPRETING THE TIMING CHART:

1. COLUMN ADDRESS is the time column address becomes valid.
2. The $\overline{CAS}$, $\overline{EAACK}$, $\overline{LAACK}$ and $\overline{XACK}$ outputs are not issued during refresh.
3. $\overline{XACK}$—high is reset asynchronously by command going inactive and not by a clock edge.
4. $\overline{EAACK}$ is used in synchronous mode, $\overline{LAACK}$ and $\overline{XACK}$ in asynchronous mode.
5. ADDRESS-Row is the clock edge where the 82C08 A0 switches from current column address to the next row address.
6. If a cycle is inhibited by PCTL = 1 (Multibus I/F mode) then $\overline{CAS}$ is not activated during write cycle and $\overline{XACK}$ is not activated in either read or write cycles.

*Column addresses switch to row addresses for next memory cycle. The row address buffer is transparent following this clock edge. 'TRAH' specification is guaranteed as per data sheet.

82C08—DRAM Interface Parameter Equations

Several DRAM parameters, but not all, are a direct function of 82C08 timings, and the equations for these parameters are given in the following tables. The following is a list of those DRAM parameters which have NOT been included in the following tables, with an explanation for their exclusion.

WRITE CYCLE

tDS: system-dependent parameter.
tDH: system-dependent parameter.
tDHR: system-dependent parameter.

READ, WRITE REFRESH CYCLES

tRAC: response parameter.
tCAC: response parameter.
tREF: See "Refresh Period Options".
tCRP: must be met only if CAS-only cycles, which do not occur with 82C08, exist.
tRAH: See "A.C. Characteristics"
tRCD: See "A.C. Characteristics"
tASC: See "A.C. Characteristics"
tASR: See "A.C. Characteristics"
tOFF: response parameter.

Table 10. Programmable Timings

Read and Refresh Cycles

Parameter	C2-Slow Cycle	C0-Fast Cycle	C1-Fast Cycle	Notes
tRP	2TCLCL-T27	3TCLCL-T27	3TCLCL-T27	1
tCPN	1.5TCLCL-T34	3TCLCL-T34	2TCLCL-T34	1, 5
tCPN	2.5TCLCL-T34	4TCLCL-T34	3TCLCL-T34	1, 4
tRSH	2TCLCL-T32	2TCLCL-T32	3TCLCL-T32	1
tCSH	3TCLCL-T25	4TCLCL-T25	6TCLCL-T25	1, 5
tCSH	2TCLCL+T34(min)-T25	3TCLCL-T25	5TCLCL-T25	1, 4
tCAH	3TCLCL-T32	2TCLCL-T32	3TCLCL-T32	1
tAR	3TCLCL-T25	3TCLCL-T25	4TCLCL-T25	1
tT	3/30	3/30	3/30	2
tRC	4TCLCL	6TCLCL	7TCLCL	1
tRAS	2TCLCL-T25	3TCLCL-T25	4TCLCL-T25	1
tCAS	3TCLCL-T32	3TCLCL-T32	5TCLCL-T32	1, 5
tCAS	2TCLCL+T34(min)-T32	2TCLCL-T25	4TCLCL-T32	1, 4
tRCS	TCLCL + T32(min)-T35 + TBUF	TCLCL-T35 + TBUF	2TCLCL-T35 + TBUF	1
tRCH	TCLCL + T36(min)-T34 + TBUF	TCLCL-T34 + TBUF	2TCLCL-T34 + TBUF	1

Write Cycles

Parameter	C2-Slow Cycle	C0-Fast Cycle	C1-Fast Cycle	Notes
tRP	2TCLCL-T27	3TCLCL-T27	3TCLCL-T27	1
tCPN	2TCLCL-T34	4TCLCL-T34	4TCLCL-T34	1
tRSH	TCLCL-T32	2TCLCL-T32	3TCLCL-T32	1
tCSH	3TCLCL-T25	4TCLCL-T25	5TCLCL-T25	1
tCAH	2TCLCL-T32	TCLCL-T32	2TCLCL-T32	1
tAR	3TCLCL-T25	3TCLCL-T25	4TCLCL-T25	1
tT	3/30	3/30	3/30	2
tRC	4TCLCL	7TCLCL	8TCLCL	1
tRAS	2TCLCL-T25	4TCLCL-T25	5TCLCL-T25	1
tCAS	2TCLCL-T32 + TBUF	2TCLCL-T32	3TCLCL-T32	1
tWCH	TCLCL-T32 + TBUF	2TCLCL-T32 + TBUF	3TCLCL-T32 + TBUF	1,3
tWCR	2TCLCL-T25 + TBUF	4TCLCL-T25 + TBUF	5TCLCL-T25 + TBUF	1,3
tWP	2TCLCL-T36-TBUF	3TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tRWL	2TCLCL-T36-TBUF	3TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tCWL	3TCLCL-T36-TBUF	3TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tWCS	TCLCL+T36-T31-TBUF	TCLCL−T36-TBUF	TCLCL−T36-TBUF	1

NOTES:
1. Minimum.
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization.
4. For synchronous mode only.
5. For asynchronous mode only.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	−0°C to +70°C
Storage Temperature	−65°C to +150°C
Voltage On Any Pin With Respect to Ground	−0.5V to +7V
Power Dissipation	0.5W

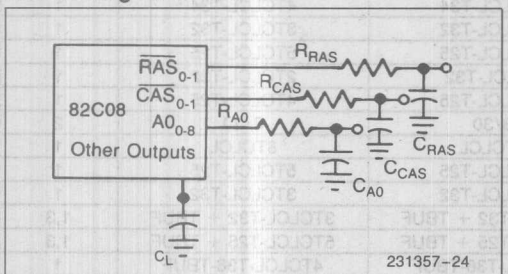
**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$; $V_{CC} = 5.0V \pm 10\%$; $V_{SS} = GND$

Symbol	Parameter	Min	Max	Units	Comments
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.6		V	(Note 1)
I_{CC}	Supply Current		10 + 2f	mA	(Note 3)
I_{LI}	Input Leakage Current		±10	μA	$0V \le V_{IN} \le V_{CC}$
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.8	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		20	pF	fc = 1 MHz(6)
V_{OHPD}	RAS Output High Power Down	V_{CC} − 0.5		V	(Note 2)
I_{PD}	Power Down Supply Current	—	5.0	mA	(Note 5)
I_{SB}	Standby Current	—	2.0	mA	(Note 4)

NOTE:
1. I_{OL} = 5 mA and I_{OH} = −0.32 mA WE: I_{OL} = 8 mA
2. $\overline{RAS}$ Output voltage during power down.
3. Typical value. Where f is freq. in MHz.
for CMOS: V_{IL} max = 0.5V; V_{IH} min = (V_{CC} − 0.5V)
for TTL: I_{CC} will be higher by 30 mA
4. Measured at V_{IL} = 0V and V_{IH} = V_{CC} with no loads connected.
5. IPD = 1 mA at 32 KHz with no loads connected.
6. Sampled, not 100% tested. T_A = 25°C.

A.C. Testing Load Circuit

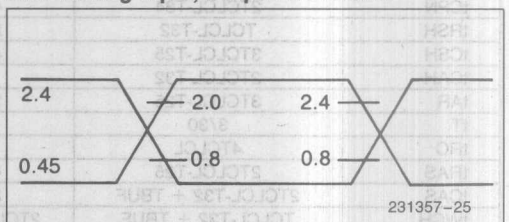


231357-24

R_{RAS} = 39Ω
R_{CAS} = 39Ω
R_{A0} = 22Ω

C_{RAS} = 150 pF
C_{CAS} = 150 pF
C_{A0} = 200 pF
C_L = 50 pF

A.C. Testing Input, Output Waveform



231357-25

A.C. Testing inputs (except clock) are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0" (clock* is driven at 4.0V and 0.45V for Logic "1" and "0" respectively). Timing measurements are made at 2.4V for Logic "1" and 0.8V for Logic "0".

* also PDCLK

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ±10%, V_{SS} = 0V)

Measurements made with respect to RAS_{0-1}, CAS_{0-1}, AO_{0-8}, are at +2.4V and 0.8V CLK at 3V, 1V. All other pins are measured at 2.0V and 0.8V. All times are ns unless otherwise indicated. Testing done with specified test load.

Ref	Symbol	Parameter	Min	Max	Units	Notes
CLOCK AND PROGRAMMING						
	tF	Clock Fall Time		12	ns	3
	tR	Clock Rise Time		12	ns	3
1	TCLCL	Clock Period				
		82C08-20	50	250	ns	1
		82C08-16	62.5	250	ns	1
		82C08-10	100	500	ns	2
		82C08-8	125	500	ns	2
2	TCL	Clock Low Time				
		82C08-20	12	230	ns	1
		82C08-16	15	230	ns	1
		82C08-10	44		ns	2
		82C08-8	TCLCL/2-12		ns	2
3	TCH	Clock High Time				
		82C08-20	16	230	ns	1
		82C08-16	20	230	ns	1
		82C08-10	44		ns	2
		82C08-8	TCLCL/3+2		ns	2
4	TRTVCL	Reset to CLK ↓ Setup	40		ns	4
5	TRTH	Reset Pulse Width	4TCLCL		ns	
6	TPGVRTL	PCTL, PDI, RFRQ to RESET ↓ Setup	125		ns	5
7	TRTLPGX	PCTL, RFRQ to RESET ↓ Hold	10		ns	
8	TCLPC	PCLK from CLK ↓ Delay		45	ns	
9	TPDICL	PDI to CLK ↓ Setup	60		ns	
10	TCLPDX	PDI to CLK ↓ Hold	40		ns	6
SYNCHRONOUS μP PORT INTERFACE						
11	TPEVCL	PE to CLK ↓ Setup	30			2
12	TKVCL	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK ↓ Setup	20		ns	1
13	TCLKX	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK ↓ Hold	0		ns	
14	TKVCH	RD, WR, PCTL to CLK ↑ Setup	20		ns	2

A.C. CHARACTERISTICS (Continued)

Ref	Symbol	Parameter	Min	Max	Units	Notes
ASYNCHRONOUS μP PORT INTERFACE						
15	TRWVCL	$\overline{RD}$, $\overline{WR}$ to CLK ↓ Setup	20		ns	8.9
16	TRWL	$\overline{RD}$, $\overline{WR}$ Pulse Width	2TCLCL+30		ns	
17	TRWLPEV	$\overline{PE}$ from $\overline{RD}$, $\overline{WR}$ ↓ Delay				
		CFS=1		TCLCL-20	ns	1
		CFS=0		TCLCL-30	ns	2
18	TRWLPEX	$\overline{PE}$ to $\overline{RD}$, $\overline{WR}$ ↓ Hold	2TCLCL+30		ns	
19	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$ ↓ Delay		TCLCL-30	ns	2
20	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$ ↓ Hold	2TCLCL+30		ns	2
21	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$ ↓ Delay		2TCLCL-20	ns	1
22	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$ ↓ Hold	3TCLCL+30		ns	1
RAM INTERFACE						
23	TAVCL	AL, AH, BS to CLK ↓ Set-up	35+tASR		ns	2
		82C08-20	50+tASR		ns	
		82C08-16	45+tASR		ns	
24	TCLAX	AL, AH, BS to CLK ↓ Hold	0		ns	
25	TCLRSL	RAS ↓ from CLK ↓ Delay		25	ns	1
				35	ns	2
				60	ns	24
26	TRCD	$\overline{RAS}$ to $\overline{CAS}$ Delay				
		CFS=1	TCLCL-25		ns	1, 14
		CFS=0	30		ns	23
		CFS=0	TCLCL/2-30		ns	2, 11, 14
		CFS=0	60		ns	2, 12, 14
27	TCLRSH	$\overline{RAS}$ ↑ from CLK ↓ Delay		25	ns	
				60	ns	24

A.C. CHARACTERISTICS (Continued)

Ref	Symbol	Parameter	Min	Max	Units	Notes
RAM INTERFACE (Continued)						
28	TRAH	CFS = 1	18			1, 13, 15
		CFS = 0	TCLCL/4-10		ns	2, 11, 15
		CFS = 0	18		ns	23
29	TASR	Row A0 $\overline{RAS}\downarrow$ Setup				10, 16
30	TASC	Column A0 to $\overline{CAS}\downarrow$ Setup				
		CFS = 1	2		ns	1, 13, 17, 18
		CFS = 0	5		ns	2, 13, 17, 18
		CFS = 0	5		ns	23
31	TCAH	Column A0 to $\overline{CAS}$ Hold	(See DRAM Interface Tables)			
32	TCLCSL	$\overline{CAS}\downarrow$ from CLK $\downarrow$ Delay				
		CFS = 0	TCLCL/4 + 30	TCLCL/1.8 + 56	ns	2, 26
		CFS = 0	50	105	ns	23, 26
		CFS = 0	8	35	ns	2, 23, 27
		CFS = 1		35	ns	1
34	TCLCSH	$\overline{CAS}\uparrow$ from CLK $\downarrow$ Delay		50	ns	
			TCLCL/4	$\frac{TCLCL}{3.2}$ + 50	ns	22
35	TCLWL	WE $\downarrow$ from CLK $\downarrow$ Delay		35	ns	
36	TCLWH	WE $\uparrow$ from CLK $\downarrow$ Delay				
		CFS = 0	TCLCL/4 + 30	TCLCL/1.8 + 53		2
		CFS = 1		35	ns	1
		CFS = 0	50	100	ns	23
37	TCLTKL	$\overline{XACK}\downarrow$ from CLK $\downarrow$ Delay		35	ns	
38	TRWLTKH	$\overline{XACK}\uparrow$ from $\overline{RD}\uparrow$, $\overline{WR}\uparrow$ Delay		50	ns	
39	TCLAKL	$\overline{AACK}\downarrow$ from $\overline{CLK}\downarrow$ Delay		35	ns	
40	TCLAKH	$\overline{AACK}\uparrow$ from CLK $\downarrow$ Delay		50	ns	
49	TARH	Column Address to $\overline{RAS}\uparrow$ Hold Time	2			1

A.C. CHARACTERISTICS (Continued)

Ref	Symbol	Parameter	Min	Max	Units	Notes
REFRESH REQUEST						
41	TRFVCL	RFRQ to CLK ↓ Setup	20		ns	
42	TCLRFX	RFRQ to CLK ↓ Hold	10		ns	
43	TFRFH	Failsafe RFRQ Pulse Width	TCLCL + 30		ns	19
44	TRFXCL	Single RFRQ Inactive to CLK ↓ Setup	20		ns	20
45	TBRFH	Burst RFRQ Pulse Width	2TCLCL + 30		ns	19
46	TPDDVCL	PDD Setup Time	20		ns	24, 25
47	TPDHRFX	RFRQ Valid after PDD Active	4TCLCL + 20			24
48	TRFVPDH	RFRQ Setup Time to PDD Active	20			24

The following RC loading is assumed:

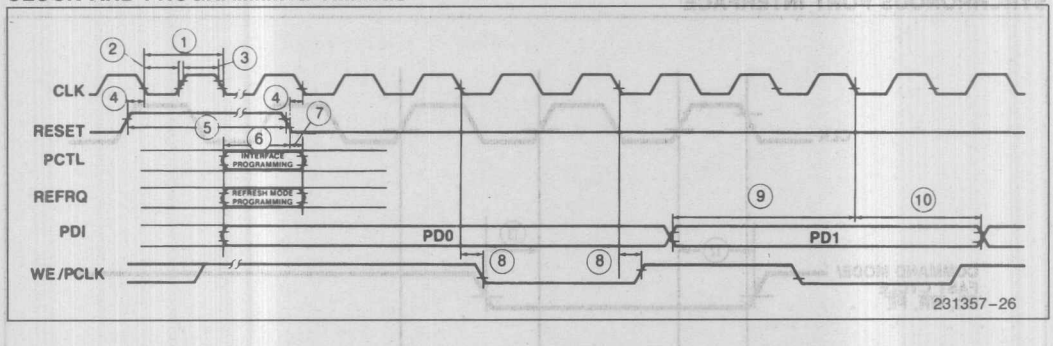
$A0_{0-8}$	R = 22Ω	C = 200 pF
$\overline{RAS}_{0-1}$, $\overline{CAS}_{0-1}$	R = 39Ω	C = 150 pF
$\overline{AACK}$, WE/PCLK		C = 50 pF

NOTES:

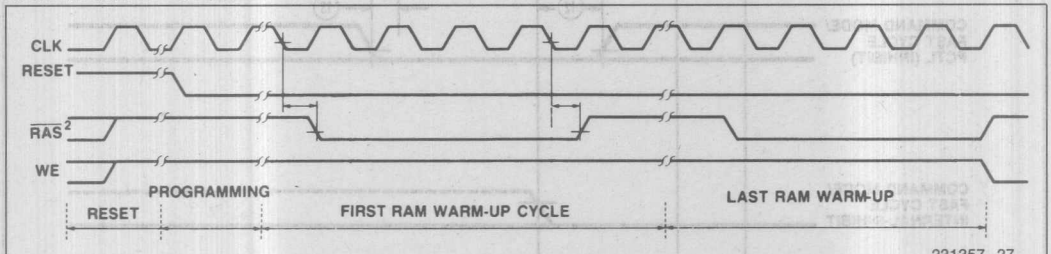
1. Specification when programmed in the Fast Cycle processor mode (iAPX 286 mode). 82C08-20, -16.
2. Specification when programmed in the Slow Cycle processor mode (iAPX 186 mode). 82C08-10, 82C08-8.
3. tR and tF are referenced from the 3.5V and 1.0V levels.
4. RESET is internally synchronized to CLK. Hence a set-up time is required only to guarantee its recognition at a particular clock edge.
5. The first programming bit (PD0) is also sampled by RESET going low.
6. TCLPDX is guaranteed if programming data is shifted using PCLK.
8. TRWVCL is not required for an asynchronous command except to guarantee its recognition at a particular clock edge.
9. Valid when programmed in either Fast or Slow Cycle mode.
10. tASR is a user specified parameter and its value should be added accordingly to TAVCL.
11. When programmed in Slow Cycle mode and 125 ns ≤ TCLCL < 200 ns.
12. When programmed in Slow Cycle mode and 200 ns ≤ TCLCL.
13. Specification for Test Load conditions.
14. tRCD (actual) = tRCD (specification) + 0.06 (ΔC_{RAS}) − 0.06(ΔC_{CAS}) where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)
15. tRAH (actual) = tRAH (specification) + 0.06 (ΔC_{RAS}) − 0.022 (ΔC_{A0}) where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)
16. tASR (actual) = tASR (specification) + 0.06 (ΔC_{A0}) − 0.025 (ΔC_{RAS}) where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)
17. tASC (actual) = tASC (specification) + 0.06 (ΔC_{A0}) − 0.025 (ΔC_{CAS}) where ΔC (test load) − C (actual) in pF. (These are first order approximations.)
18. tASC is a function of clock frequency and thus varies with changes in frequency. A minimum value is specified.
19. TFRFH and TBRFH pertain to asynchronous operation only.
20. Single RFRQ should be supplied synchronously to avoid burst refresh.
22. CFS = 0, synchronous mode, Read cycle.
23. For 10 MHz Slow Cycle only.
24. Power down mode.
25. PDD is internally synchronized. A setup time is required only to guarantee its recognition at a particular clock edge.
26. Slow Cycle Read only.
27. Slow Cycle Write only.

WAVEFORMS

CLOCK AND PROGRAMMING TIMINGS

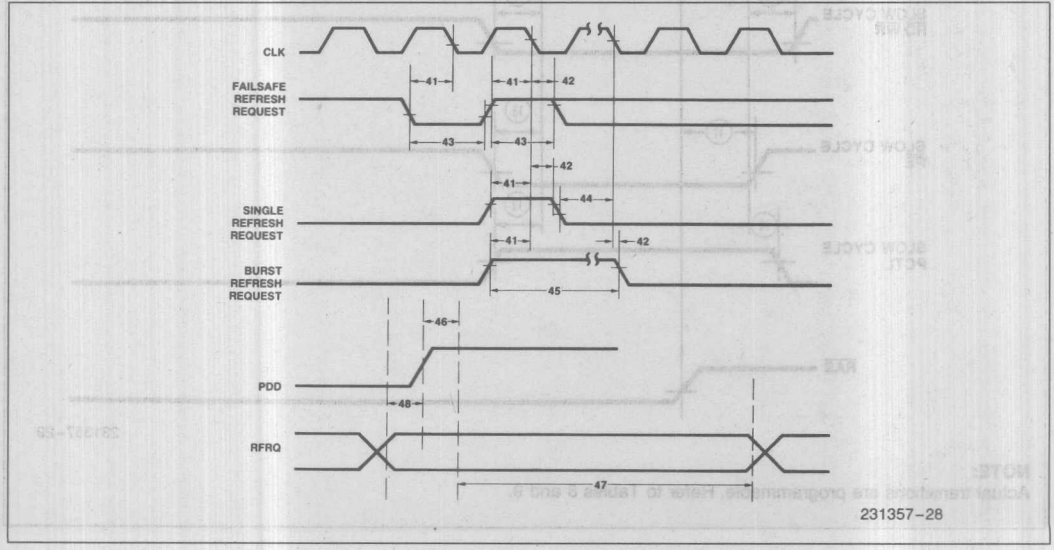


RAM WARM-UP CYCLES



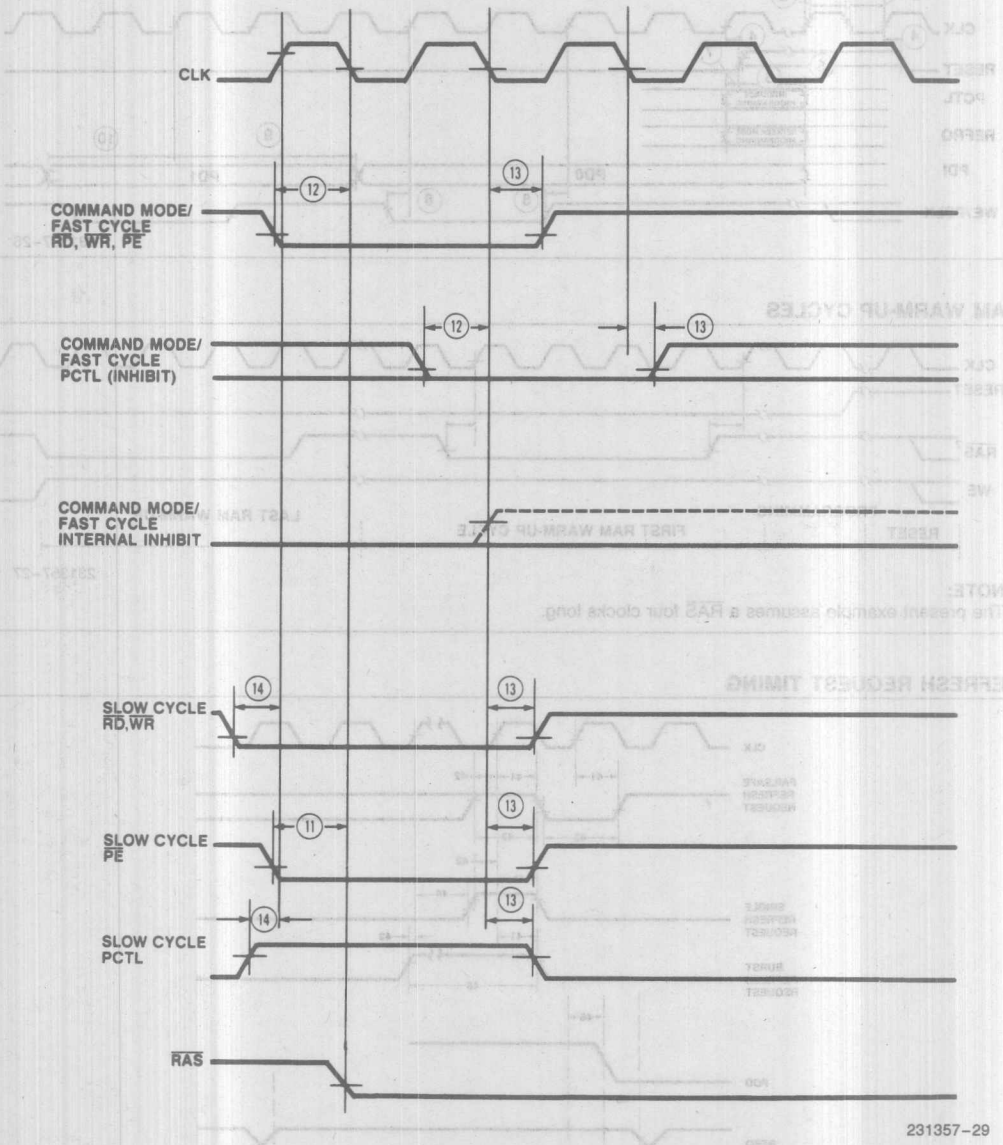
NOTE:
The present example assumes a $\overline{RAS}$ four clocks long.

REFRESH REQUEST TIMING



WAVEFORMS (Continued)

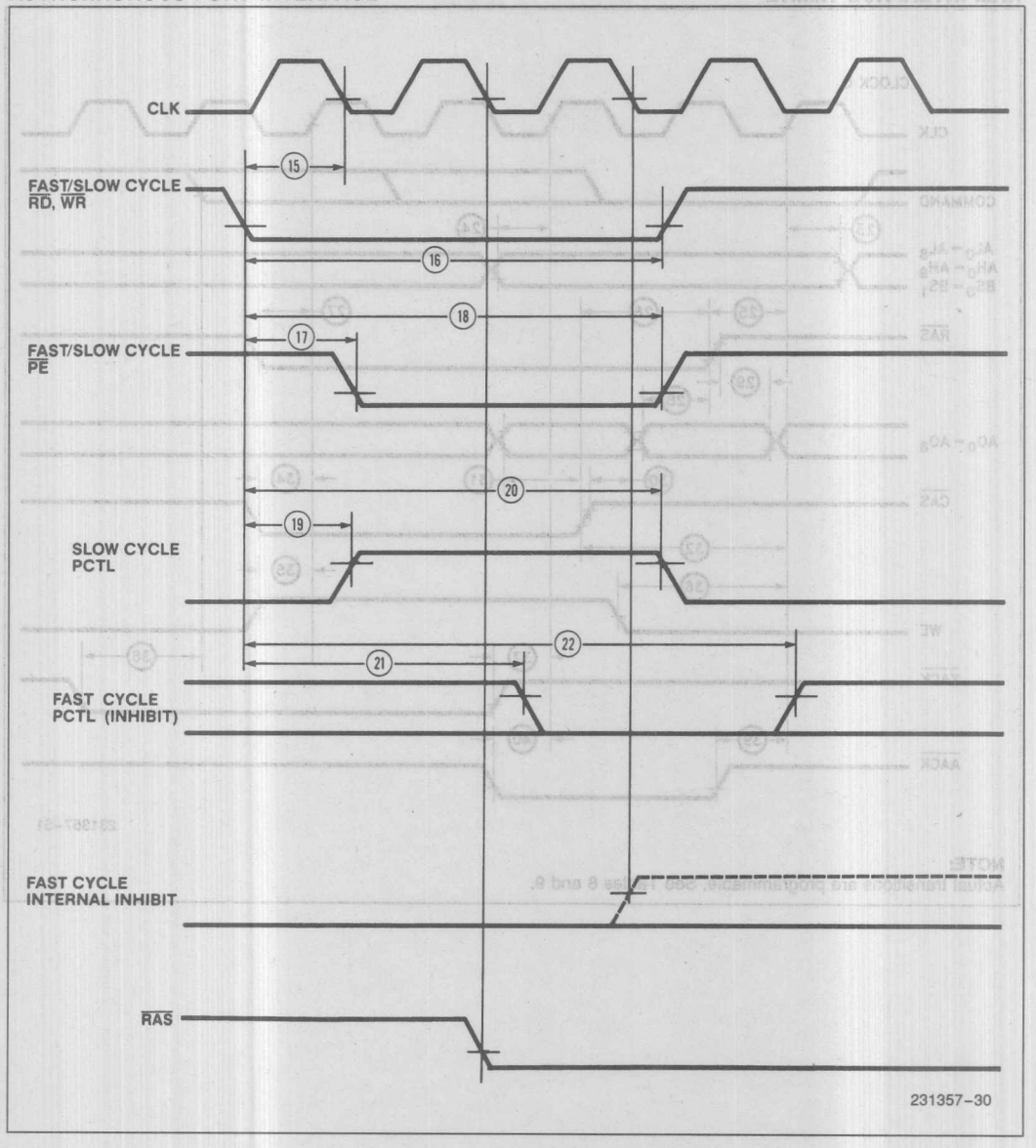
SYNCHRONOUS PORT INTERFACE



NOTE:
Actual transitions are programmable. Refer to Tables 8 and 9.

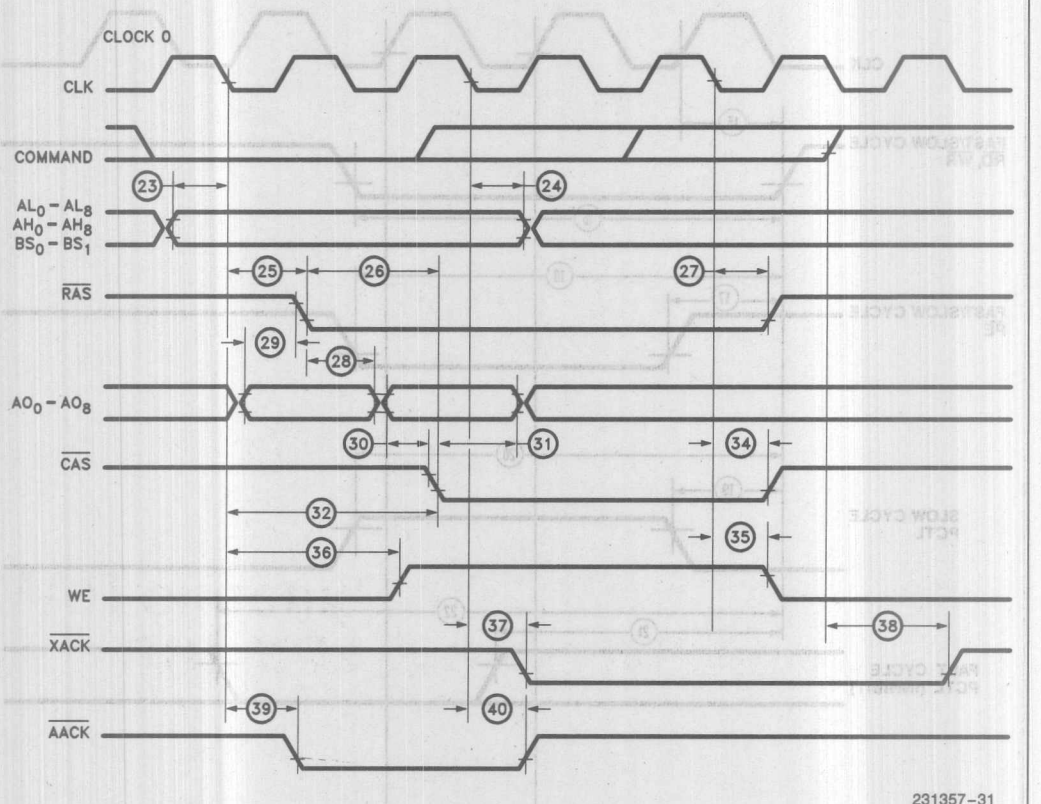
WAVEFORMS (Continued)

ASYNCHRONOUS PORT INTERFACE



WAVEFORMS (Continued)

RAM INTERFACE TIMING



NOTE:
Actual transitions are programmable. See Tables 8 and 9.

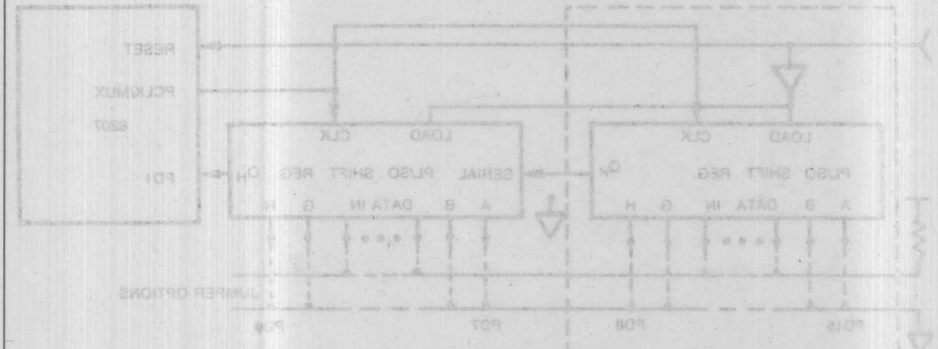
intel

APPLICATION NOTE

AP-167

November 1986

Interfacing the 8207 Dynamic RAM Controller to the iAPX 186



JIM SLEEZER
APPLICATION ENGINEER

Order Number: 230809-001

INTRODUCTION

Most microprocessor based workstation designs today use large amounts of DRAM for program storage. A drawback to DRAMs is the many critical timings that must be met. This control function could easily equal the area of the DRAM array if implemented with discrete logic.

The VLSI 8207 Advanced Dynamic RAM Controller (ADRC) performs complete DRAM timing and control. This includes the normal RAM 8 warm-up cycles, various refresh cycles and frequencies, address multiplexing, and address strobe timing. The 8207's system interface and RAM timing and control are programmable to permit it to be used in most applications.

Integrating all of the above functions (plus a dual port and error correcting interfaces) allows the user to realize significant cost savings over discrete logic. For example, comparing the 8207 to the iSBC012B 512K byte RAM board (where the DRAM control is done entirely with TTL), an 8207 design saved board space 3 in^2 vs 10 in^2); required less power (420 mA vs 1220 mA); and generated less heat. Moreover, design time was reduced, and increased margins were achieved due to less skewing of critical timings. This comparison is based on a single port design and did not include the 8207's RAM warm-up, dual-port and error correcting features. If these features were fully implemented, there would be no change to the 8207 figures, listed above, while the TTL figures would easily double.

This Application Note will illustrate an iAPX design with the 8207 controlling the dynamic RAM array. The reader should be familiar with the 82097 data sheet, the 80186 data sheet, and a RAM data sheet*.

DESIGN GOALS

The main objective of this design is for the 80186 to run with no wait states with a Dynamic RAM array. The design uses one port of the 8207. The dual port and error correcting interfaces of the 8207 are covered in separate Application Notes.

The size of the RAM array is 4 banks of 64K RAMs or 512K bytes. The memory is to be interfaced locally to the 80186.

USING THE 8207

The three areas to be considered when designing in the 8207 are:

- 8207 programming logic
- Microprocessor interface
- RAM array

8207 Programming

The 8207 requires up to two 74LS165 shift registers for programming. This design needs one 8 bit shift register, as shown in Figure 1. The 16 bits in the Program Data Word are set as shown in Figure 2. Refresh is done internally, so the REFRQ input must be tied high. The memory commands are iAPX 86 status, so the timing

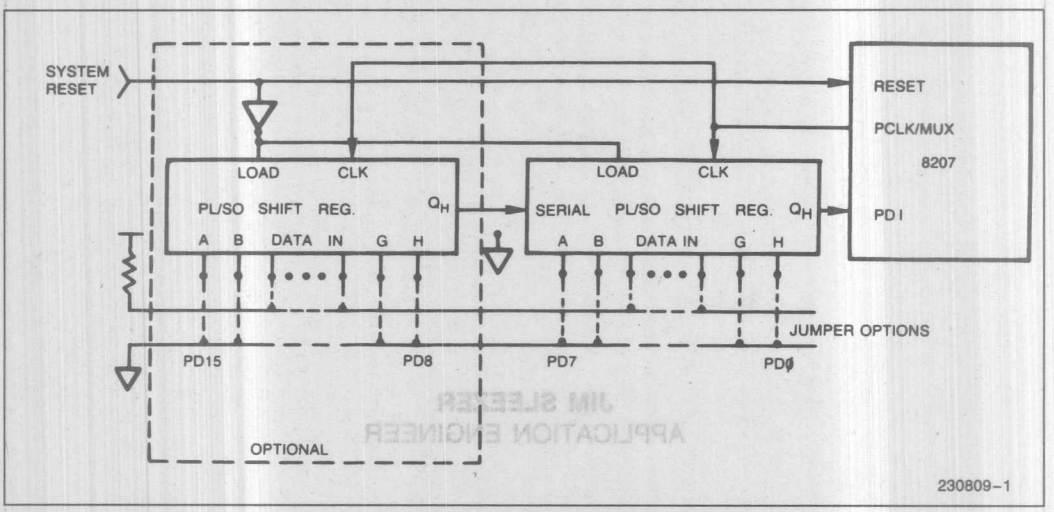
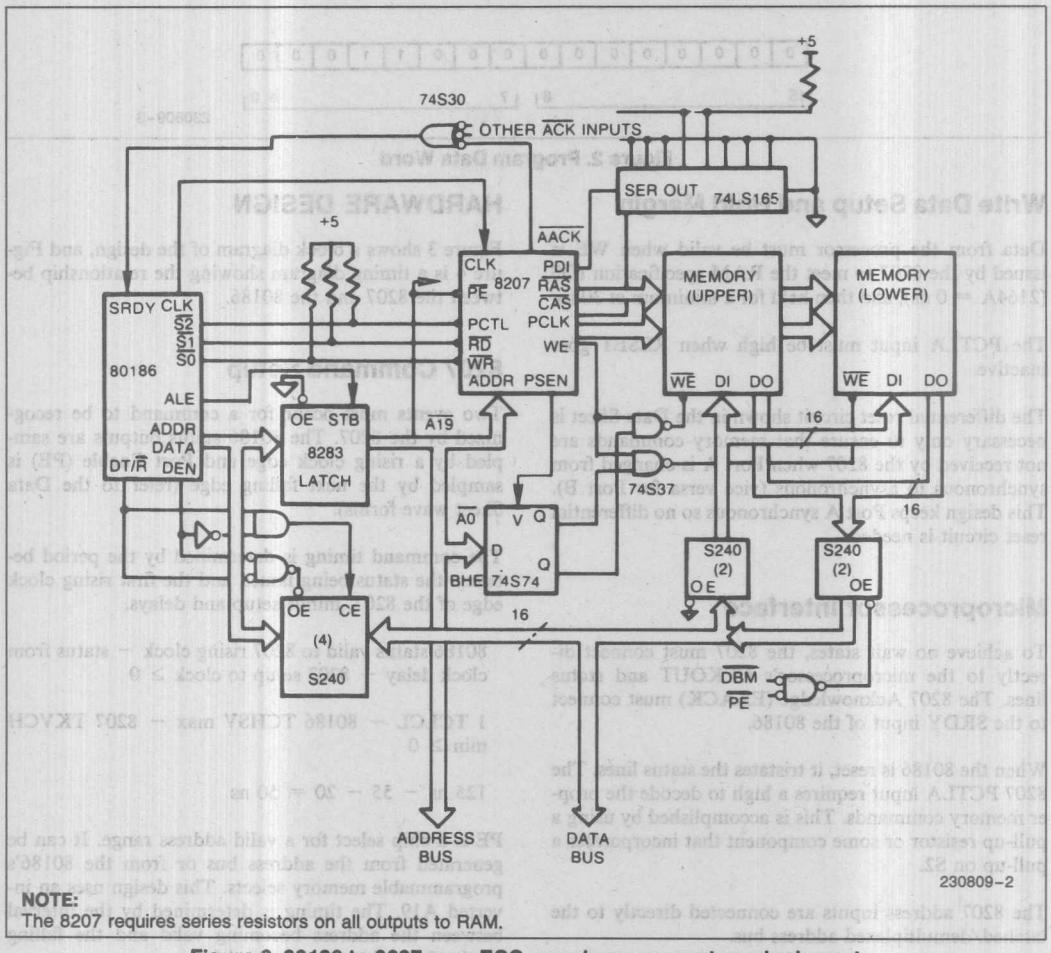


Figure 1. 8207 Programming Shift Registers

*All RAM references in this Application Note are based on Intel's 2164A 64K Dynamic RAM.



NOTE:
The 8207 requires series resistors on all outputs to RAM.

Figure 3. 80186 to 8207, non-ECC, synchronous system single port

of EAACK will always guarantee 2 clocks of address hold time from RAS.

Acknowledge Setup Time

The margin between the 8207 issuing EAACK and the 80186 ready input for no wait states minus delays from clock edges, logic delays, and setup time is calculated as follows.

1 clock − 8207 TCLAKL max − 74S30 t_{PLH} @ 15 pf − 80186 TSRYCL ≥ 0

125 ns − 35 − 22 − 35 = 32 ns

Read Access Margin

The 8207 starts a memory cycle on the falling clock edge between the 80186's T1 and T2. Data must be valid within 2 clocks. Valid data from the RAMs is based upon the CAS access period minus buffer, clock, setup requirements.

2 TCLCL − 8207 TCLCSL @ 150 pF (t34) − DRAM tCAC − 74S240 propagation delay @ 50 pF − additional bus loading delay (250 pF)(1) − 74S240 delay @ 50 pF − 80186 TDVCL ≥ 0

250 ns − 122 − 85 − 7 − 7 − 7 − 20 = 2 ns

NOTE:
(1) 74STTL logic derated by 0.05 ns/pF. 74STTL buffers (240, 37) derated by 0.025 ns/pF.

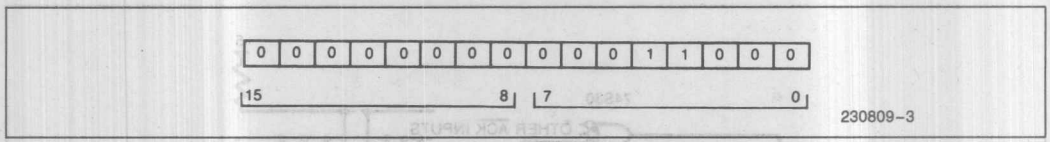


Figure 2. Program Data Word

Write Data Setup and Hold Margin

Data from the processor must be valid when WE is issued by the 8207 to meet the RAM specification tDS (2164A = 0 ns), and then held for a minimum of 30 ns.

The PCTLA input must be high when RESET goes inactive.

The differential reset circuit shown in the Data Sheet is necessary only to ensure that memory commands are not received by the 8207 when Port A is changed from synchronous to asynchronous (vice versa for Port B). This design keeps Port A synchronous so no differential reset circuit is needed.

Microprocessor Interface

To achieve no wait states, the 8207 must connect directly to the microprocessor's CLKOUT and status lines. The 8207 Acknowledge (EAACK) must connect to the SRDY input of the 80186.

When the 80186 is reset, it tristates the status lines. The 8207 PCTLA input requires a high to decode the proper memory commands. This is accomplished by using a pull-up resistor or some component that incorporates a pull-up on S2.

The 8207 address inputs are connected directly to the latched/demultiplexed address bus.

RAM Array

The 8207 provides complete control of all RAM timings, warm up cycles, and refresh cycles. All write cycles are "late writes." During write cycles, the data out lines go active. This requires separate data in/out lines in the RAM array.

To operate the 80186 with no wait states, it is necessary to choose sufficiently fast DRAMs. The 150 ns version of the 2164A allows operating the 80186 at 8 MHz, and the 200 ns version up to 7 MHz.

HARDWARE DESIGN

Figure 3 shows a block diagram of the design, and Figure 4 is a timing diagram showing the relationship between the 8207 and the 80186.

8207 Command Setup

Two events must occur for a command to be recognized by the 8207. The 80186 status outputs are sampled by a rising clock edge and Port Enable (PE) is sampled by the next falling edge (refer to the Data Sheet wave forms).

The command timing is determined by the period between the status being issued and the first rising clock edge of the 8207, minus setup and delays.

80186 status valid to 8207 rising clock − status from clock delay − 8207 setup to clock ≥ 0

1 TCLCL − 80186 TCHSV max − 8207 TKVCH min ≥ 0

125 ns − 55 − 20 = 50 ns

PE is a chip select for a valid address range. It can be generated from the address bus or from the 80186's programmable memory selects. This design uses an inverted A19. The timing is determined by the interval between the address becoming valid and the falling clock edge, minus setup and delays.

80186 address valid to 8207 falling clock edge − 80186 address from clock delay − 8283 latch delays − 8207 PE setup ≥ 0

1 TCLCL − 80186 TCLAV max − 8283 IVOV @ 300 pF − 8207 TPEVCL ≥ 0

125 ns − 44 − 22 − 30 − 29 ns

The hold times are 0 ns and are met.

Address Setup

For an 81086 design, the 8207 requires the address to be stable before RAS goes active, and to remain stable for 2 clocks. Unused 8207 address inputs should be tied to V_{CC}.

tASR is a RAM specification. If it is greater than zero, this must be added to the address setup time of the 807. Address setup is the interval between addresses being issued and RAS going active, minus appropriate delays.

80186 address valid to 8207 RAS active − 80186 address from clock delay − bus delays − (8207 setup + RAM t_{ASR}) ≥ 0

TCLCL + 8207 TCLRSL min @ 150 pf(1) − 80186 TCLAV max − 8283 IVOV max @ 300 pf − (8207 TAVCL min + DRAM t_{ASR}) ≥ 0

125 ns + 0 − 44 − 22 − (35 + 0) = 24 ns

The address hold time of 2 clocks + 0 ns is always met, since the addresses are latched by the 8282/3. Even when the processor is in wait states (for refresh),

NOTE:
(1) Not specified—use 0 ns.

TCLCL + TCLCH + 8207 TCLW min(1) + 74S37 delay tPHL min @ 50 pf + additional loading (142 pf) − 81086 TCVCTV − 74S240tPZL − bus delays (250 pf) − 74S240 delay − 2164A tDS ≥ 0

125 + 62.5 + 0 + 6.5 + 3.5 − 70 − 15 − 7 − 7 − 0 = 98.5 ns.

The hold time, tDH, is from WE going low to the 80186 DEN going high plus buffer delays minus WE from clock delays.

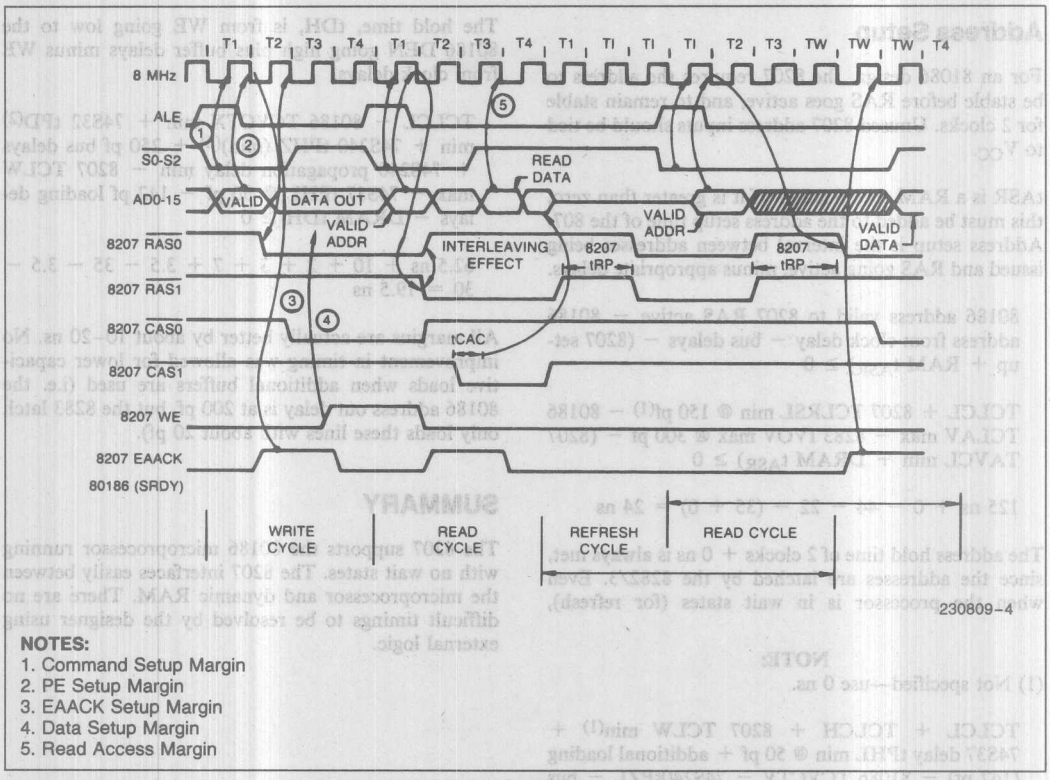
TCLCL − 80186 TCVCTX min + 74S32 tPD(2) min + 74S240 tPHZ (min)(2) + 250 pf bus delays + 74S240 propagation delay min − 8207 TCLW max − 74S37 tPHL @ 50 pf − 142 pf loading delays − DRAM tDH ≥ 0

62.5 ns + 10 + 2 + 3 + 7 + 3.5 − 35 − 3.5 − 30 = 19.5 ns

All margins are actually better by about 10–20 ns. No improvement in timing was allowed for lower capacitive loads when additional buffers are used (i.e. the 80186 address out delay is at 200 pf, but the 8283 latch only loads these lines with about 20 pf).

SUMMARY

The 8207 supports the 80186 microprocessor running with no wait states. The 8207 interfaces easily between the microprocessor and dynamic RAM. There are no difficult timings to be resolved by the designer using external logic.



NOTES:
1. Command Setup Margin
2. PE Setup Margin
3. EAACK Setup Margin
4. Data Setup Margin
5. Read Access Margin

Figure 4. 8207/80186 Timing Relationship

NOTE:
1. Not specified, use 0 ns.
2. Not specified, use one half of typical value.

APPLICATION NOTE

AP-168

November 1986

Interfacing the 8207 Advanced Dynamic RAM Controller to the iAPX 286

JIM SLEEZER
APPLICATION ENGINEER

Order Number: 230862-001

INTRODUCTION

The 80286 high speed microprocessor pushes microprocessor based systems to new performance levels. However, its high speed bus requires special design considerations to utilize that performance. Interfacing the 80286 to a dynamic RAM array require many timings to be analyzed, refresh cycle effects on bus timing examined, minimum and maximum signal widths noted, and the list continues.

The 8207 Advanced Dynamic RAM Controller was specifically designed to solve all interfacing issues for the 80286, provide complete control and timing for the DRAM array, plus achieve optimum system performance. This includes the normal RAM 8 warmup cycles, various refresh cycles and frequencies, address multiplexing, and address strobe timings. The 8207 Dynamic RAM Controller's system interface and RAM timing and control are programmable to permit it to be used in most applications.

Integrating these functions (plus dual port and error correcting interfaces) allows the user to realize significant savings in both engineering design time, PC board space and product cost. For example, in comparing the 8207 to the ISBC012B 512k byte RAM board (where the DRAM timing and control is done entirely with TTL), the 8207 design saved board space (3 in^2 vs 10 in^2); used less power (420 mA vs 1220 mA); reduced the design time; and increased margins due to less skewing of timings. The comparison is based upon a single port 8207 design and does not include its RAM warm-up, dual port, error correcting, and error scrubbing or RAM interleaving features.

This Application Note will detail an 80286 and 8207 design. The reader should have read the 8207 and the 80286 data sheets, a DRAM data sheet*, and have them available for reference.

DESIGN GOALS

The main objective of this design is to run the RAM array without wait states, to maximize the 80286's performance, and to use as little board space as possible. The 80286 will interface synchronously to Port A of the 8207 and the 8207 will control 512k bytes of RAM (4 banks using 64k DRAMs). The dual port and error correcting features of the 8207 are covered in separate Application Notes.

8207 INTERFACE

The 8207 Memory design can be subdivided into three sections:

- Programming the 8207.
- The 82086/8207 interface.
- The Dynamic RAM array.

Programming the 8207

The RAM timing is configured via the 16 bit program word that the 8207 shifts-in when reset. This can require two 74LS165 shift registers to provide complete DRAM configurability. The 8207 defaults to the configuration shown in Table 1 when PDI is connected to ground. This design does not need the flexibility the shift registers would allow since standard 8207/80286 clock frequencies, DRAM speeds and refresh rates are used. Table 1 details the 8207/80286 configuration and Table 10 in the Data Sheet identifies "CO" as the configuration of the 8207 all timings will be referenced to (80286 mode at 16 MHz using fast RAMs = CO).

Table 1. Default Non-ECC Programming, PD1 Pin (57) Tied to Ground

Port A is Synchronous (EAACKA and XACKA)
Port B is Asynchronous (LAACKB and XACKB)
Fast-cycle Processor Interface (10 or 16 MHz)
Fast RAM 100/120 ns RAM
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

The 8207 will accept 80286 status inputs when the PCTLA pin is sampled low at reset. This pin is not necessary for an 80286 design (besides programming) and is tied to ground.

Refresh is the final option to be programmed. If the Refresh pin is sampled high at reset, an internal timer

*All RAM references in this Application Note are based upon Intel's CMOS 51C64-12 64k Dynamic RAM. Any DRAM with similar timings will function. Refer to section 4.4.

is enabled, and if low at reset, this timer is disabled. The first method is the easiest to implement, so the RFRQ pin is tied to V_{CC}.

The differential reset circuit shown in the Data Sheet is necessary only to ensure that memory commands are not received by the 8207 when Port A is changed from synchronous to asynchronous (vice versa for Port B). This design keeps Port A synchronous so no differential reset circuit is needed.

RAM Array

The 8207 completely controls all RAM timings, warm-up cycles, and refresh cycles. To determine if a particular RAM will work with the 8207, calculate the margins provided by the 8207 (Table 15, 16—8207 Data Sheet) and ensure they are greater than the RAM requirement. An additional consideration is the access times of the RAMs. The access time of the system is dependent upon the number of data buffers between the 80286 and the DRAMs. To operate the 80286 at zero wait states requires access times of 100–120 ns. Slower RAMs can be used (150 ns) by either adding a wait state (programming the 8207 for "C1") or reducing the clock frequency (to 14.9 MHz approximately and maintaining the CO configuration).

All write cycles are "late writes" and the data out lines of the RAM will go active. This will require separate data in and out lines in the RAM array. Another consideration for the RAM array is the proper layout of the RAM, and impedance matching resistors on the 8207 outputs. Proper layout is covered in Intel's RAM Data Sheets and Application Notes.

Microprocessor Array

To achieve no wait state operation, the 8207's clock input must be connected to the 80286's clock input. The EAACK (early acknowledge) output of the 8207 must connect to the SRDY input of the 82284. The 8207's address inputs connect directly to the 80286 address outputs and the addresses are latched internally. This latch is strobed by an internal signal with the same timing as LEN (which is for dual port 82086 designs). Figure 2 shows the timing relationship between LEN and the 80286.

LEN will fall from high to low, which latches the bus address internally, when a valid command is received. LEN can go high in two clock cycles if the RAM cycle started (RAS going low) at the same time LEN went low. If the 8207 is doing a refresh cycle, the 80286 will be put into wait states until the memory cycle can start. LEN will then go high two clocks after RAS starts, since addresses are no longer needed for the current RAM cycle. Thus the low period of LEN could be much longer than listed in the Data Sheet.

DESIGNING THE HARDWARE

Figure 1 shows a detailed block diagram of the design and Figure 2 shows the timing relationship between the 8207 and the 80286.

The following analysis of six parameters will confirm that the design will work. These six system parameters are generally considered the most important in any microprocessor—Dynamic RAM design.

8207 Command Setup Margin

Two events must occur for the 8207 to start a memory cycle. Either RD or WR active (low) and PE must be low when the 8207 samples these pins on a falling clock edge. If PE is not valid at the same clock edge that samples RD or WR active, the memory cycle will be aborted and no acknowledge will be issued.

The command setup time is based upon the status being valid at the first falling clock edge.

80286 status valid to 8207 falling clock − 80286 status from clock delay − 8207 command setup to clock ≤ 0

TCLCL − 80286 t12 (max) − 8207 TKVCL (min) ≤ 0

62.5 − 40 ns − 20 ns = 2.5 ns

PE is decoded from the address bus and must be set up to the same falling clock edge that recognizes the RD, WR inputs. This margin is determined from the clock edge that issues the address and the clock edge that will recognize RD or WR, minus decoding logic delays.

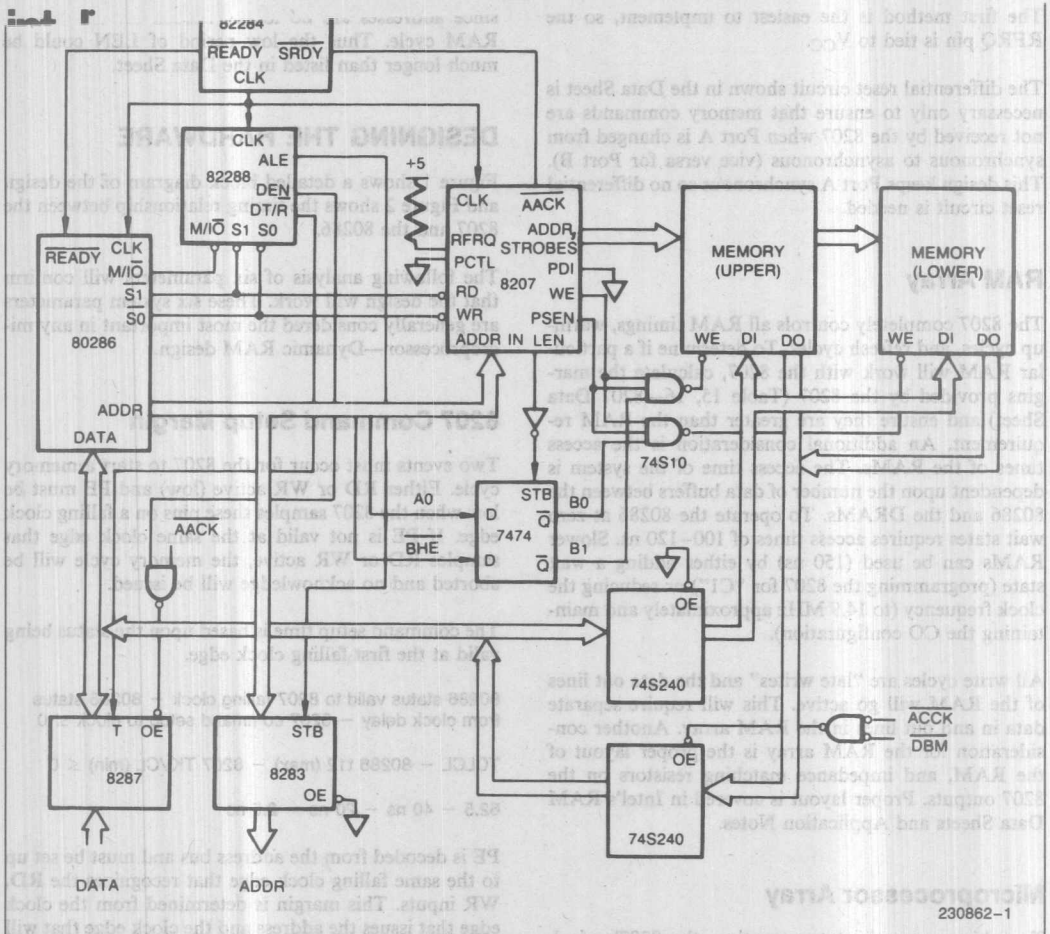
There are 2 clocks between addresses being issued by the 80286 and PE being sampled by the 8207. Then the 80286 address delay from the clock edge and decoding logic delays are subtracted from this interval. This margin must be greater than 0.

2TCLCL − 80286 t13 (max) − 8207 TPEVCL (min) ≤ 0

125 − 60 − 30 = 35 ns

The address decode logic must use no more than 35 ns (and less is better). Figure 3 shows an easy implementation which uses a maximum of 12 ns.

The 8207 requires a zero ns hold time and is always met.



NOTE:
The 8207 requires series resistors on all outputs.

Figure 1. 80286 to 8207, non-ECC, Synchronous System Single Port

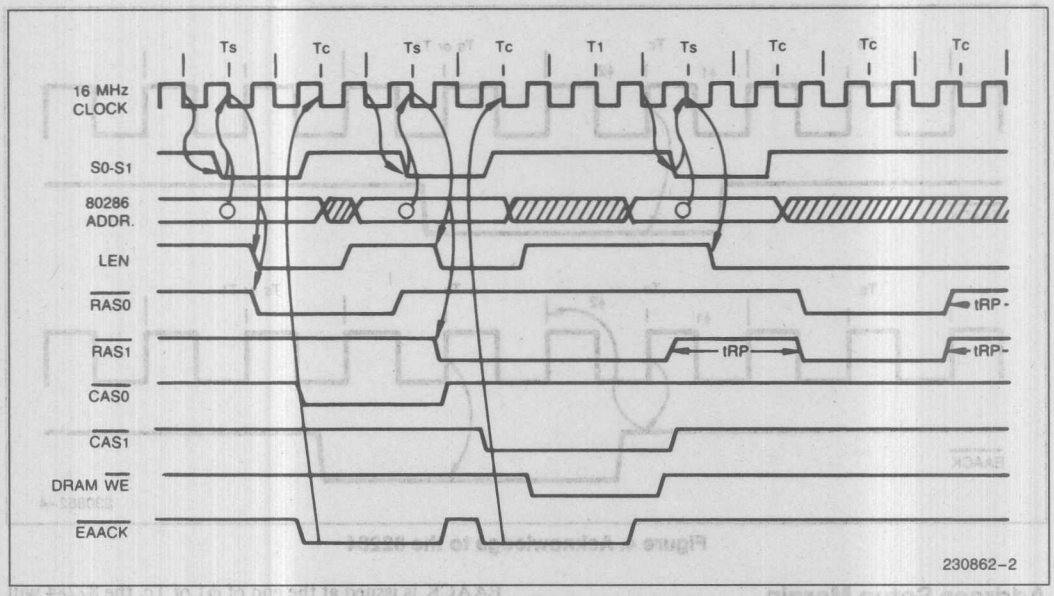


Figure 2. 80286/8207 Timing—"CO"

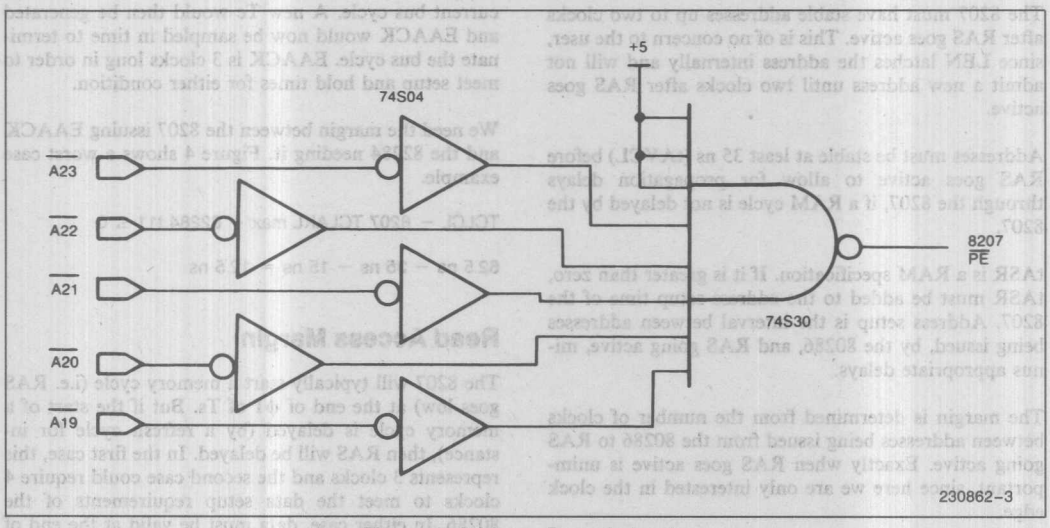


Figure 3. Address Decode Logic

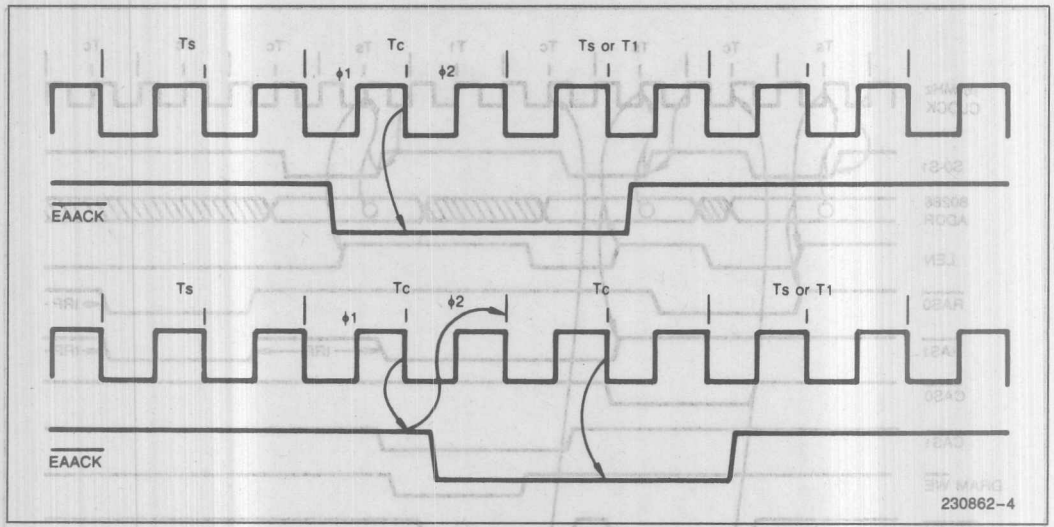


Figure 4. Acknowledge to the 82284

Address Setup Margin

The 8207 must have stable addresses up to two clocks after RAS goes active. This is of no concern to the user, since LEN latches the address internally and will not admit a new address until two clocks after RAS goes active.

Addresses must be stable at least 35 ns (tAVCL) before RAS goes active to allow for propagation delays through the 8207, if a RAM cycle is not delayed by the 8207.

tASR is a RAM specification. If it is greater than zero, tASR must be added to the address setup time of the 8207. Address setup is the interval between addresses being issued, by the 80286, and RAS going active, minus appropriate delays.

The margin is determined from the number of clocks between addresses being issued from the 80286 to RAS going active. Exactly when RAS goes active is unimportant, since here we are only interested in the clock edge.

2TCLCL − 80286 t13 (max) − 8207 TAVCL (min) ≤ 0

125 ns − 60 ns − 35 ns = 30 ns

Acknowledge Setup Margin

The 8207 acknowledge (EAACK) can be issued at any point in the 80286 bus cycle (end of $\phi 1$ or $\phi 2$ of Ts or Tc). If EAACK is issued at the end of $\phi 2$ (Ts or Tc), the 80286 will complete the current bus cycle. If EAACK is issued at the end of $\phi 1$ of Tc, the 82284 will not generate READY to the 80286 in time to end the current bus cycle. A new Tc would then be generated and EAACK would now be sampled in time to terminate the bus cycle. EAACK is 3 clocks long in order to meet setup and hold times for either condition.

We need the margin between the 8207 issuing EAACK and the 82284 needing it. Figure 4 shows a worst case example.

TCLCL − 8207 TCLAKL max − 82284 t11 ≤ 0

62.5 ns − 35 ns − 15 ns = 12.5 ns

Read Access Margin

The 8207 will typically start a memory cycle (i.e. RAS goes low) at the end of $\phi 1$ of Ts. But if the start of a memory cycle is delayed (by a refresh cycle for instance), then RAS will be delayed. In the first case, this represents 3 clocks and the second case could require 4 clocks to meet the data setup requirements of the 80286. In either case, data must be valid at the end of Tc. The 8207 holds CAS active long enough to ensure valid data is received by the 80286 in either case.

DRAMs specify two access times, RAS access (tRAC) and CAS access (tCAC) Both access periods must be calculated and the one with the least margin used. Also the number of data buffers should be kept to a minimum. Too many buffers would require either faster (more expensive) DRAMs, or a reduction in the performance of the CPU (by adding wait states).

RAS Access Margin

3TCLCL − 8207 TCLRSL max @ 150 pF − DRAM tRAC − 74S240 propagation delay max @ 50 pF − 80286 t8 ≤ 0

187.5 ns − 35 ns − 120 ns − 7 ns − 10 ns = 15.5 ns

CAS Access Margin

2TCLCL − 820-7 TCLCSL max @ 150 pF − DRAM tCAA (or tCAC − 74S240 tplh max @ 50 pF − 80286 t8) ≤ 0

125 ns − 35 ns − 60 ns − 7 ns − 10 ns = 13 ns

By solving each equation for tRAC and tCAC, the speed requirement of the RAM can be determined.

DRAM tRAC = 3 TCLCL − 8207 TCLRSL − 74S240 tplh − 82086 t8 = 135.5 ns

DRAM tCAC = 2 TCLCL − 8207 TCLCSL − 74S240 tplh − 80286 t8 = 73 ns

NOTES:
1. Not specified. Assume no delay for worst case analysis.
2. STTL derated by 0.05 ns/pF.

So any DRAM that has a RAS access period less than 135 ns, a CAS access period less than 73 ns, and meets all requirements in the DRAM Interface Timing (Table 15, 16—8207 Data Sheet), will work.

Write Data Setup and Hold Margin

Write data from the processor must be valid when the 8207 issues WE to meet the DRAM specification tDS and then held to meet the tDH requirement. Some write cycles will be byte writes and the information to determine which byte is decoded from A0 and BHE/. Since the 80286's address bus is pipelined, these two signals can change before the RAM cycle starts, hence they must be latched by LEN. PSEN is used in the WE term to shorten the WE pulse. Its use is not essential.

Data must be set up to the falling edge of WE, since WE occurs after CAS. The 2 clocks between valid write data and WE going active (at the RAM's) minus propagation delays determines the margin.

2 TCLCL − 80286 t14 (max) @ 100 pF − 74S240 tplh + 8207 TCLW (min)[1] + 74S10 tphl @ 192 pF[2] − DRAM tDS = 0

125 ns − 50 ns − 7 ns + 0 ns + 14 ns − 0 ns = 82 ns

The timing of the 8207's acknowledge is such that data will be kept valid by the 80286, for more than two clocks after WE goes active. This easily meets all RAM tDH specifications.

SUMMARY

The 8207 complements the 80286's performance and high integration with its own performance, integration and ease of use. No critical timings or logic design has been left to the designer. The 80286/8207 combination allows users to realize maximum performance from their simpler design.

RAS Access Margin

$$2TCLCL - 8207\ TCLRSL\ max\ @\ 180\ pF - DRAM\ tRAC - 74S260\ propagation\ delay\ max\ @\ 50\ pF - 80286\ tS \leq 0$$

$$187.5\ ns - 35\ ns - 120\ ns - 7\ ns - 10\ ns = 15.5\ ns$$

CAS Access Margin

$$2TCLCL - 8207\ TCLCSL\ max\ @\ 180\ pF - DRAM\ tCAA\ (or\ tCAC) - 74S240\ tplh\ max\ @\ 50\ pF - 80286\ tS) \leq 0$$

$$125\ ns - 35\ ns - 80\ ns - 7\ ns - 10\ ns = 13\ ns$$

By solving each equation for tRAC and tCAC, the speed requirement of the RAM can be determined.

$$DRAM\ tRAC = 3\ TCLCL - 8207\ TCLRSL - 74S240\ tplh - 80286\ tS = 156.5\ ns$$

$$DRAM\ tCAC = 2\ TCLCL - 8207\ TCLCSL - 74S240\ tplh - 80286\ tS = 73\ ns$$

NOTES:

1. Not specified. Assume no delay for worst case analysis.
2. STTL derated by 0.05 ns/pF.

So any DRAM that has a RAS access period less than 15.5 ns, a CAS access period less than 73 ns, and meets all requirements in the DRAM Interface Timing (Table 15, 16—8207 Data Sheet), will work.

Write Data Setup and Hold Margin

Write data from the processor must be valid when the 8207 issues WE to meet the DRAM specification tDS and then hold to meet the tDH requirement. Some write cycles will be byte writes and the information to determine which byte is decoded from A0 and BHE/. Since the 80286's address bus is pipelined, these two signals can change before the RAM cycle starts, hence they must be latched by LEN. PSIZN is used in the WE term to shorten the WE pulse. Its use is not essential.

Data must be set up to the falling edge of WE, since WE occurs after CAS. The 2 clocks between valid write data and WE going active (for the RAM's) minus propagation delays determines the margin.

$$2\ TCLCL - 80286\ tr4\ (max)\ @\ 100\ pF - 74S240\ tplh + 8207\ TCLW\ (min)^1 + 74S10\ tplh\ @\ 182\ pF - DRAM\ tDS = 0$$

$$125\ ns - 80\ ns - 7\ ns + 0\ ns + 14\ ns - 0\ ns = 82\ ns$$

This timing of the 8207's adjustable is such that data will be kept valid by the 80286, for more than two clocks after WE goes active. This easily meets all RAM tDH specifications.

SUMMARY

The 8207 complements the 80286's performance and high integration with its own performance, integration and ease of use. No critical timings or logic design has been left to the designer. The 80286/8207 combination allows users to realize maximum performance from their simpler design.

2 Support Peripherals

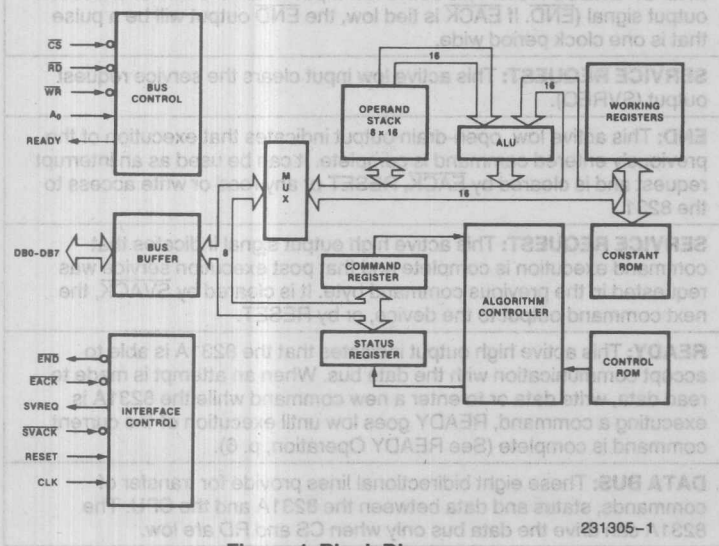
8231A
ARITHMETIC PROCESSING UNIT

- **Fixed Point Single and Double Precision (16/32 Bit)**
- **Floating Point Single Precision (32 Bit)**
- **Binary Data Formats**
- **Add, Subtract, Multiply and Divide**
- **Trignometric and Inverse Trigonometric Functions**
- **Square Roots, Logarithms, Exponentiation**
- **Float to Fixed and Fixed to Float Conversions**
- **Stack Oriented Operand Storage**
- **Compatible with all Intel and most other Microprocessor Families**
- **Direct Memory Access or Programmed I/O Data Transfers**
- **End of Execution Signal**
- **General Purpose 8-Bit Data Bus Interface**
- **Standard 24 Pin Package**
- **+12V and +5V Power Supplies**
- **Advanced N-Channel Silicon Gate HMOS Technology**

The Intel® 8231A Arithmetic Processing Unit (APU) is a monolithic HMOS LSI device that provides high performance fixed and floating point arithmetic and floating point trigonometric operations. It may be used to enhance the mathematical capability of a wide variety of processor-oriented systems. Chebyshev polynomials are used in the implementation of the APU algorithms.

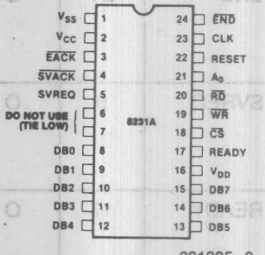
All transfers, including operand, result, status and command information, take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack and commands are issued to perform operations on the data and the stack. Results are then available to be retrieved from the stack.

Transfers to and from the APU may be handled by the associated processor using conventional programmed I/O, or may be handled by a direct memory access controller for improved performance. Upon completion of each command, the APU issues an end of execution signal that may be used as an interrupt by the CPU to help coordinate program execution.



231305-1

Figure 1. Block Diagram



231305-2

Figure 2. Pin Configuration

September 1987
Order Number: 231305-002

Table 1. Pin Description

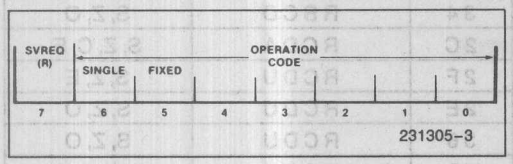
Symbol	Pin No.	Type	Name and Function
V_{CC}	2		**POWER:** +5V power supply.
V_{DD}	16		**POWER:** +12V power supply.
V_{SS}	1		**GROUND.**
CLK	23	I	**CLOCK:** An external, TTL compatible, timing source is applied to the CLK pin.
RESET	22	I	**RESET:** The active high reset signal provides initialization for the chip. RESET also terminates any operation in progress. RESET clears the status register and places the 8231A into the idle state. Stack contents and command registers are not affected (5 clock cycles).
$\overline{CS}$	18	I	**CHIP SELECT:** $\overline{CS}$ is an active low input signal which selects the 8231A and enables communication with the data bus.
A_0	21	I	**ADDRESS:** In conjunction with the $\overline{RD}$ and WR signals, the A_0 control line establishes the type of communication that is to be performed with the 8231A as shown below:

A_0	$\overline{RD}$	$\overline{WR}$	Function
0	1	0	Enter data byte into stack
0	0	1	Read data byte from stack
1	1	0	Enter command
1	0	1	Read status

$\overline{RD}$	20	I	**READ:** This active low input indicates that data or status is to be read from the 8231A if CS is low.
$\overline{WR}$	19	I	**WRITE:** This active low input indicates that data or a command is to be written into the 8231A if CS is low.
$\overline{EACK}$	3	I	**END OF EXECUTION:** This active low input clears the end of execution output signal ($\overline{END}$. If $\overline{EACK}$ is tied low, the $\overline{END}$ output will be a pulse that is one clock period wide.
$\overline{SVACK}$	4	I	**SERVICE REQUEST:** This active low input clears the service request output (SVREQ).
$\overline{END}$	24	O	**END:** This active low, open-drain output indicates that execution of the previously entered command is complete. It can be used as an interrupt request and is cleared by $\overline{EACK}$, RESET or any read or write access to the 8231.
SVREQ	5	O	**SERVICE REQUEST:** This active high output signal indicates that command execution is complete and that post execution service was requested in the previous command byte. It is cleared by $\overline{SVACK}$, the next command output to the device, or by RESET.
READY	17	O	**READY:** This active high output indicates that the 8231A is able to accept communication with the data bus. When an attempt is made to read data, write data or to enter a new command while the 8231A is executing a command, READY goes low until execution of the current command is complete (See READY Operation, p. 6).
DB0–DB7	8-15	I/O	**DATA BUS:** These eight bidirectional lines provide for transfer of commands, status and data between the 8231A and the CPU. The 8231A can drive the data bus only when CS and RD are low.

COMMAND STRUCTURE

Each command entered into the 8231A consists of a single 8-bit byte having the format illustrated below:



Bits 0–4 select the operation to be performed as shown in the table. Bits 5–6 select the data format appropriate to the selected operation. If bit 5 is a 1, a fixed point data format is specified. If bit 5 is 0, floating point format is specified. Bit 6 selects the precision of the data to be operated upon by fixed point commands only (if bit 5 = 0, bit 6 must be 0). If bit 6 is a 1, single-precision (16-bit) operands are assumed. If bit 6 is a 0, double-precision (32-bit) operands are indicated. Results are undefined for all illegal combinations of bits in the command byte. Bit 7 indicates whether a service request is to be issued after the command is executed. If bit 7 is a 1, the service request output (SVREQ) will go high at the conclusion of the command and will remain high until reset by a low level on the service acknowledge pin ($\overline{SVACK}$) or until completion of execution of the succeeding command where service request (bit 7) is 0. Each command issued to the 8231A requests post execution service based upon the state of bit 7 in the command byte. When bit 7 is a 0, SVREQ remains low.

Table 2. 32-Bit Floating Point Instructions

Instruction	Description	Hex(1) Code	Stack Contents(2) After Execution A B C D	Status Flags(4) Affected
ACOS	Inverse Cosine of A	0 6	R U U U	S, Z, E
ASIN	Inverse Sine of A	0 5	R U U U	S, Z, E
ATAN	Inverse Tangent of A	0 7	R B U U	S, Z
CHSF	Sign Change of A	1 5	R B C D	S, Z
COS	Cosine of A (radians)	0 3	R B U U	S, Z
EXP	e^A Function	0 A	R B U U	S, Z, E
FADD	Add A and B	1 0	R C D U	S, Z, E
FDIV	Divide B by A	1 3	R C D U	S, Z, E
FLTD	32-Bit Integer to Floating Point Conversion	1 C	R B C U	S, Z
FLTS	16-Bit Integer to Floating Point Conversion	1 D	R B C U	S, Z
FMUL	Multiply A and B	1 2	R C D U	S, Z, E
FSUB	Subtract A from B	1 1	R C D U	S, Z, E
LOG	Common Logarithm (base 10) of A	0 8	R B U U	S, Z, E
LN	Natural Logarithm of A	0 9	R B U U	S, Z, E
POPF	Stack Pop	1 8	B C D A	S, Z
PTOF	Stack Push	1 7	A A B C	S, Z
PUPI	Push π onto Stack	1 A	R A B C	S, Z
PWR	B^A Power Function	0 B	R C U U	S, Z, E
SIN	Sine of A (radians)	0 2	R B U U	S, Z
SQRT	Square Root of A	0 1	R B C U	S, Z, E
TAN	Tangent of A (radians)	0 4	R B U U	S, Z, E
XCHF	Exchange A and B	1 9	B A C D	S, Z

Table 3. 32-Bit Integer Instructions

Instruction	Description	Hex[1] Code	Stack Contents[2] After Execution A B C D	Status Flags[4] Affected
CHSD	Sign Change of A	3 4	R B C D	S, Z, O
DADD	Add A and B	2 C	R C D A	S, Z, C, E
DDIV	Divide B by A	2 F	R C D U	S, Z, E
DMUL	Multiply A and B (R = lower 32-bits)	2 E	R C D U	S, Z, O
DMUU	Multiply A and B (R = upper 32-bits)	3 6	R C D U	S, Z, O
DSUB	Subtract A from B	2 D	R C D A	S, Z, C, O
FIXD	Floating Point to Integer Conversion	1 E	R B C U	S, Z, O
POPD	Stack Pop	3 8	B C D A	S, Z
PTOD	Stack Push	3 7	A A B C	S, Z
XCHD	Exchange A and B	3 9	B A C D	S, Z

Table 4. 16-Bit Integer Instructions

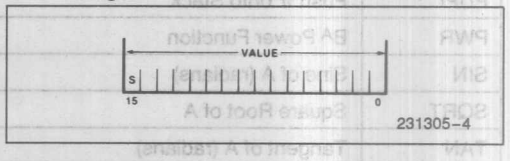
Instruction	Description	Hex[1] Code	Stack Contents[3] After Execution A_U A_L B_U B_L C_U C_L D_U D_L	Status Flags[4] Affected
CHSS	Change Sign of A_U	7 4	R A_L B_U B_L C_U C_L D_U D_L	S, Z, O
FIXS	Floating Point to Integer Conversion	1 F	R B_U B_L C_U C_L U U U	S, Z, O
POPS	Stack Pop	7 8	A_L B_U B_L C_U C_L D_U D_L A_U	S, Z
PTOS	Stack Push	7 7	A_U A_U A_L B_U B_L C_U C_L D_U	S, Z
SADD	Add A_U and A_L	6 C	R B_U B_L C_U C_L D_U D_L A_U	S, Z, C, E
SDIV	Divide A_L by A_U	6 F	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SMUL	Multiply A_L by A_U (R = lower 16-bits)	6 E	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SMUU	Multiply A_L by A_U (R = upper 16-bits)	7 6	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SSUB	Subtract A_U from A_L	6 D	R B_U B_L C_U C_L D_U D_L A_U	S, Z, C, E
XCHS	Exchange A_U and A_L	7 9	A_L A_L B_U B_L C_U C_L D_U D_L	S, Z
NOP	No Operation	0 0	A_U A_L B_U B_L C_U C_L D_U D_L	

NOTES:
1. In the hex code column, SVREQ is a 0.
2. The stack initially is composed of four 32-bit numbers (A, B, C, D). A is equivalent to Top Of Stack (TOS) and B is Next On Stack (NOS). Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A, B, C, or D).
3. The stack initially is composed of eight 16-bit numbers (A_U, A_L, B_U, B_L, C_U, C_L, D_U, D_L). A_U is the TOS and A_L is NOS. Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A_U, A_L, B_U, B_L . . .).
4. Nomenclature: Sign (S); Zero (Z); Overflow (O); Carry (C); Error Code Field (E).

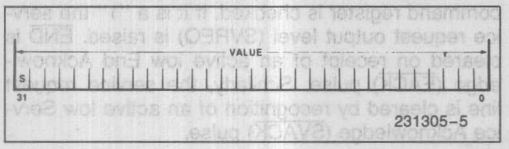
DATA FORMATS

The 8231A arithmetic processing unit handles operands in both fixed point and floating point formats. Fixed point operands may be represented in either single (16-bit operands) or double precision (32-bit operands), and are always represented as binary, two's complement values.

Single Precision Fixed Point Format



Double Precision Fixed Point Format



The sign (positive or negative) of the operand is located in the most significant bit (MSB). Positive values are represented by a sign bit of zero (S = 0). Negative values are represented by the two's complement of the corresponding positive value with a sign bit equal to 1 (S = 1). The range of values that may be accommodated by each of these formats is −32,768 to +32,767 for single precision and −2,147,483,648 to +2,147,483,647 for double precision.

Floating point binary values are represented in a format that permits arithmetic to be performed in a fashion analogous to operations with decimal values expressed in scientific notation.

$$(5.83 \times 10^{2})\ (8.16 \times 10^{1}) = (4.75728 \times 10^{4})$$

In the decimal system, data may be expressed as values between 0 and 10 times 10 raised to a power that effectively shifts the implied decimal point right or left the number of places necessary to express the result in conventional form (e.g., 47,572.8). The value-portion of the data is called the mantissa. The exponent may be either negative or positive.

The concept of floating point notation has both a gain and a loss associated with it. The gain is the ability to represent the significant digits of data with values spanning a large dynamic range limited only by the capacity of the exponent field. For example, in decimal notation in the exponent field is two digits wide, and the mantissa is five digits, a range of values (positive or negative) from 1.0000×10^{-99} to $9.9999 \times 10^{+99}$ can be accommodated. The loss is that only the significant digits of the value can be represented. Thus there is no distinction in this representation between the values 123451 and 123452, for example since each would be expressed as: 1.2345×10^{5}. The sixth digit has been discarded. In most applications where the dynamic range of values to be represented in large, the loss of significance, and hence accuracy of results, is a minor consideration. For greater precision a fixed point format could be chosen, although with a loss of potential dynamic range.

The 8231A is a binary arithmetic processor and requires that floating point data be represented by a fractional mantissa value between 0.5 and 1 multiplied by 2 raised to an appropriate power. This is expressed as follows:

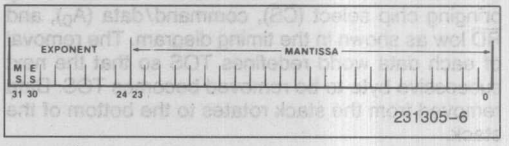
$$\text{value} = \text{mantissa} \times 2^{\text{exponent}}$$

For example, the value 100.5 expressed in this form is $0.1100\ 1001 \times 2^{7}$. The decimal equivalent of this value may be computed by summing the components (powers of two) of the mantissa and then multiplying by the exponent as shown below:

$$\begin{aligned}\text{value} &= (2^{-1} + 2^{-2} + 2^{-5} + 2^{-8}) \times 2^{7} \\ &= 0.5 + 0.25 + 0.03125 + 0.00290625) \times 128 \\ &= 0.78515625 \times 128 \\ &= 100.5\end{aligned}$$

FLOATING POINT FORMAT

The format for floating point values in the 8231A is given below. The mantissa is expressed as a 24-bit (fractional) value; the exponent is expressed as a two's complement 7-bit value having a range of −64 to +63. The most significant bit is the sign of the mantissa (0 = positive, 1 = negative), for a total of 32 bits. The binary point is assumed to be the left of the most significant mantissa bit (bit 23). All floating point data values must be normalized. Bit 23 must be equal to 1, except for the value zero, which is represented by all zeros.

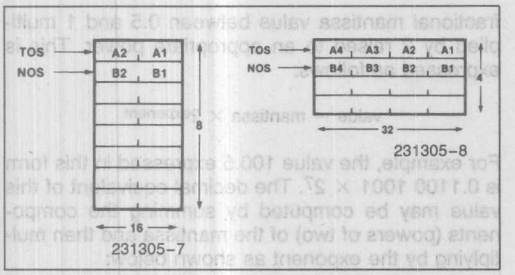


The range of values that can be represented in this format is $\pm(2.7 \times 10^{-20}$ to $9.2 \times 10^{18})$ and zero.

FUNCTIONAL DESCRIPTION

STACK CONTROL

The user interface to the 8231A includes access to an 8 level 16-bit wide data stack. Since single precision fixed point operands are 16-bits in length, eight such values may be maintained in the stack. When using double precision fixed point or floating point formats four values may be stored. The stack in these two configurations can be visualized as shown below:



Data are written onto the stack, eight bits at a time, in the order shown (A1, A2, A3, . . .). Data are removed from the stack in reverse byte order (A4, A3, A2 . . .). Data should be entered onto the stack in multiples of the number of bytes appropriate to the chosen data format.

DATA ENTRY

Data entry is accomplished by bringing the chip select ($\overline{CS}$), the command/data line (A_0), and $\overline{WR}$ low, as shown in the timing diagram. The entry of each new data word "pushes down" the previously entered data and places the new byte on the top of stack (TOS). Data on the bottom of the stack prior to a stack entry are lost.

DATA REMOVAL

Data are removed from the stack in the 8231A by bringing chip select ($\overline{CS}$), command/data (A_0), and $\overline{RD}$ low as shown in the timing diagram. The removal of each data world redefines TOS so that the next successive byte to be removed becomes TOS. Data removed from the stack rotates to the bottom of the stack.

COMMAND ENTRY

After the appropriate number of bytes of data have been entered onto the stack, a command may be issued to perform an operation on that data. Commands which require two operands for execution (e.g., add) operate on the TOS and NOS values. Single operand commands operate only on the TOS.

Commands are issued to the 8231A by bringing the chip select ($\overline{CS}$) line low, command data (A_0) line high, and $\overline{WR}$ line low as indicated by the timing diagram. After a command is issued, the CPU can continue execution of its program concurrently with the 8231A command execution.

COMMAND COMPLETION

The 8231A signals the completion of each command execution by lowering the End Execution line ($\overline{END}$). Simultaneously, the busy bit in the status register is cleared and the Service Request bit of the command register is checked. If it is a "1" the service request output level (SVREQ) is raised. $\overline{END}$ is cleared on receipt of an active low End Acknowledge ($\overline{EACK}$) pulse. Similarly, the service request line is cleared by recognition of an active low Service Acknowledge ($\overline{SVACK}$) pulse.

READY OPERATION

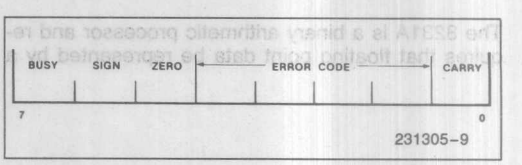
An active high ready (READY) is provided. This line is high in its quiescent state and is pulled low by the 8231A under the following conditions:

1. A previouly initiated operation is in progress (device busy) and Command Entry has been attempted. In this case, the READY line will be pulled low and remain low until completion of the current command execution. It will then go high, permitting entry of the new command.
2. A previously initiated operation is in progress and stack access has been attempted. In this case, the READY line will be pulled low, will remain in that state until execution is complete, and will then be raised to permit completion of the stack access.
3. The 8231A is not busy, and data removal has been requested. READY will be pulled low for the length of time necessary to transfer the byte from the top of stack to the interface latch, and will then go high, indicating availability of the data.
4. The 8231A is not busy, and a data entry has been requested. READY will be pulled low for the length of time required to ascertain if the preceding data byte, if any, has been written to the stack. If so READY will immediately go high. If not, READY will remain low until the interface latch is free and will then go high.
5. When a status read has been requested, READY will be pulled low for the length of time necessary to transfer the status to the interface latch, and will then be raised to permit completion of the status read. Status may be read whether or not the 8231A is busy.

When READY goes low, the APU expects the bus control signals present at the time to remain stable until READY goes high.

DEVICE STATUS

Device status is provided by means of an internal status register whose format is shown below:



Busy: Indicates that 8231A is currently executing a command (1 = Busy)

Sign: Indicates that the value on the top of stack is negative (1 = Negative)

Zero: Indicates that the value on the top of stack is zero (1 = Value is zero)

Error Code: This field contains an indication of the validity of the result of the last operation. The error codes are:

0000—No error

1000—Divide by zero

0100—Square root or log of negative number

1100—Argument of inverse sine, cosine, or e^x too large

XX10—Underflow

XX01—Overflow

Carry: Previous operation resulted in carry or borrow from most significant bit. (1 = Carry/Borrow, 0 = No Carry/No Borrow).

If the BUSY bit in the status register is a one, the other status bits are not defined; if zero, indicating not busy the operation is complete and the other status bits are defined as given above.

READ STATUS

The 8231A status register can be read by the CPU at any time (whether an operation is in progress or not) by bringing the chip select ($\overline{CS}$) low, the command/data line (A_0) high, and lowering $\overline{RD}$. The status register is then gated onto the data bus and may be input by the CPU.

EXECUTION TIMES

Timing for execution of the 8231A command set is contained below. All times are given in terms of clock cycles. Where substantial variation of execution times is possible, the minimum and maximum values are quoted; otherwise, typical values are given. Variations are data dependent.

Total execution times may require allowances for operand transfer into the APU, command execution, and result retrieval from the APU. Except for command execution, these times will be heavily influenced by the nature of the data, the control interface used, the speed of memory, the CPU used, the priority allotted to DMA and interrupt operations, the size and number of operands to be transferred, and the use of chained calculations, etc.

DERIVED FUNCTION DISCUSSION

Computer approximations of transcendental functions are often based on some form of polynomial equation, such as:

$$F(X) = A_0 + A_1X + A_2X^2 + A_3X^3 + A_4X^4 \ldots \quad (1\text{-}1)$$

Table 5. Command Execution Times

Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles
SADD	17	FADD	54-368	LN	4298-6956	POPF	12
SSUB	30	FSUB	70-370	EXP	3794-4878	XCHS	18
SMUL	84-94	FMUL	146-168	PWR	8290-12032	XCHD	26
SMUU	80-98						
SDIV	84-94	FDIV	154-184	NOP	4	XCHF	26
DADD	21	SORT	800	CHSS	23	PUPI	16
DSUB	38	SIN	4464	CHSD	27		
DMUL	194-210	COS	4118	CHSF	18		
DMUU	182-218						
DDIV	208	TAN	5754	PTOS	16		
FIXS	92-216	ASIN	7668	PTOD	20		
FIXD	100-346	ACOS	7734	PTOF	20		
FLTS	98-186	ATAN	6006	POPS	10		
FLTD	98-378	LOG	4474-7132	POPD	12		

The primary shortcoming of an approximation is this form is that it typically exhibits very large errors when the magnitude of |X| is large, although the errors are small when |X| is small. With polynomials in this form, the error distribution is markedly uneven over any arbitrary interval.

A set of approximating functions exists that not only minimizes the maximum error but also provides an even distribution of errors within the selected data representation interval. These are known as Chebyshev Polynomials and are based upon cosine functions. These functions are defined as follows:

$$T_n(X) = \text{Cos } n\theta; \text{ where } n = 0, 1, 2 \ldots \quad (1\text{-}2)$$

$$\theta = \text{Cos}^{-1} X$$

The various terms of the Chebyshev series can be computed as shown below:

$$T_0(X) = \text{Cos } (0 \times \theta) = \text{Cos } (0) = 1 \quad (1\text{-}4)$$

$$T_1(X) = \text{Cos } (\text{Cos}^{-1} X) = X \quad (1\text{-}5)$$

$$T_2(X) = \text{Cos } 2\theta = 2\text{Cos}^2\theta - 1 = 2\text{Cos}^2 (\text{Cos}^{-1} X) - 1 \quad (1\text{-}6)$$

$$= 2X^2 - 1$$

In general, the next term in the Chebyshev series can be recursively derived from the previous term as follows:

$$T_n(X) = 2X [T_n - 1(X)] - T_n - 2(X); n \geq 2 \quad (1\text{-}7)$$

Common logarithms are computed by multiplication of the natural logarithm by the conversion factor 0.43429448 and the error function is therefore the same as that for natural logarithm. The power function is realized by combination of natural log and exponential functions according to the equation:

$$X^Y = e^{y\text{Ln}x}.$$

The error for the power function is a combination of that for the logarithm and exponential functions.

Each of the derived functions is an approximation of the true function. Thus the result of a derived function will have an error. The absolute error is the difference between the function's result and the true result. A more useful measure of the function's error is relative error (absolute error/true result). This gives a measurement of the significant digits of algorithm accuracy. For the derived functions except LN, LOG, and PWR the relative error is typically 4×10^{-7}. For PWR the relative error is the summation of the EXP and LN errors, 7×10^{-7}. For LN and LOG, the absolute error is 2×10^{-7}.

APPLICATION INFORMATION

The diagram in Figure 4 shows the interface connections for the APU with operand transfers handled by an 8237 DMA controller, and CPU coordination handled by an Interrupt Controller. The APU interrupts the CPU to indicate that a command has been completed. When the performance enhancements provided by the DMA and Interrupt operations are not required, the APU interface can be simplified as shown in Figure 3. The 8231A APU is designed with a general purpose 8-bit data bus and interface control so that it can be conveniently used with any general 8-bit processor.

In many systems it will be convenient to use the microcomputer system clock to drive the APU clock input. In the case of 8080A systems it would be the ϕ2TTL signal. Its cycle time will usually fall in the range of 250 ns to 1000 ns, depending on the system speed.

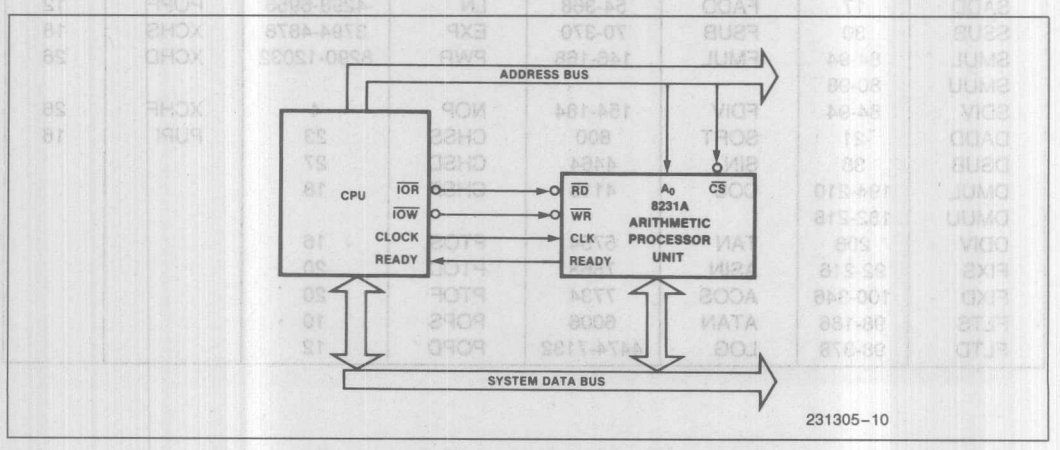


Figure 3. Minimum Configuration Example

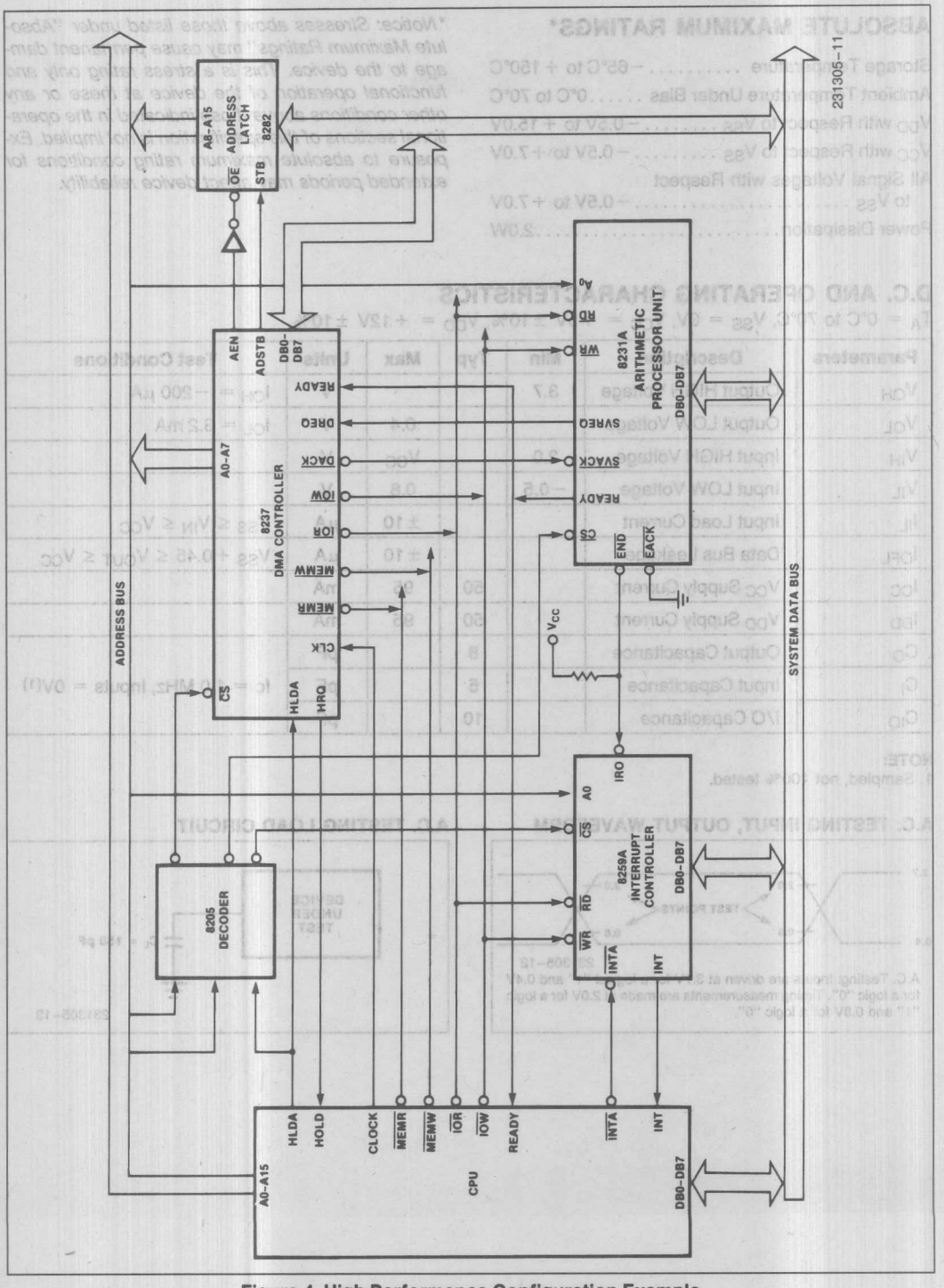


Figure 4. High Performance Configuration Example

ABSOLUTE MAXIMUM RATINGS

Storage Temperature ..... −65°C to +150°C
Ambient Temperature Under Bias ..... 0°C to 70°C
V_{DD} with Respect to V_{SS} ..... −0.5V to +15.0V
V_{CC} with Respect to V_{SS} ..... −0.5V to +7.0V
All Signal Voltages with Respect to V_{SS} ..... −0.5V to +7.0V
Power Dissipation 2.0W

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

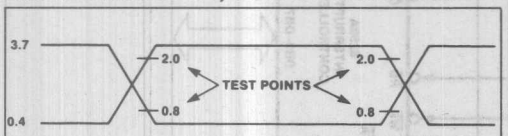
D.C. AND OPERATING CHARACTERISTICS

T_A = 0°C to 70°C, V_{SS} = 0V, V_{CC} = +5V ±10%, V_{DD} = +12V ±10%

Parameters	Description	Min	Typ	Max	Units	Test Conditions
V_{OH}	Output HIGH Voltage	3.7			V	I_{OH} = −200 μA
V_{OL}	Output LOW Voltage			0.4	V	I_{OL} = 3.2 mA
V_{IH}	Input HIGH Voltage	2.0		V_{CC}	V	
V_{IL}	Input LOW Voltage	−0.5		0.8	V	
I_{IL}	Input Load Current			±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{OFL}	Data Bus Leakage			±10	μA	V_{SS} +0.45 ≤ $V_{OUT} \le V_{CC}$
I_{CC}	V_{CC} Supply Current		50	95	mA	
I_{DD}	V_{DD} Supply Current		50	95	mA	
C_O	Output Capacitance		8		pF	fc = 1.0 MHz, Inputs = 0V(1)
C_I	Input Capacitance		5		pF	
C_{IO}	I/O Capacitance		10		pF	

NOTE:
1. Sampled, not 100% tested.

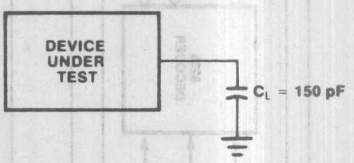
A.C. TESTING INPUT, OUTPUT WAVEFORM



231305-12

A.C. Testing: Inputs are driven at 3.7V for a logical "1" and 0.4V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

A.C. TESTING LOAD CIRCUIT



231305-13

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{SS} = 0V$, $V_{CC} = +5V \pm 10\%$, $V_{DD} = +12V \pm 10\%$

READ OPERATION

Symbol	Parameter		8231A-8 Min	8231A-8 Max	8231A Min	8231A Max	Units
t_{AR}	A_0, $\overline{CS}$ Setup to $\overline{RD}$		0		0		ns
t_{RA}	A_0, $\overline{CS}$ Hold from $\overline{RD}$		0		0		ns
t_{RY}	READY ↓ from $\overline{RD}$ ↓ Delay (Note 2)			150		100	ns
t_{YR}	Ready ↑ to $\overline{RD}$ ↑		0		0		ns
t_{RRR}	READY Pulse Width (Note 3)	Data	$3.5\ t_{CY} + 50$		$3.5\ t_{CY} + 50$		ns
		Status	$1.5\ t_{CY} + 50$		$1.5\ t_{CY} + 50$		ns
t_{RDE}	Data Bus Enable from $\overline{RD}$ ↓		50		50		ns
t_{DRY}	Data Valid to READY ↑		0		0		ns
t_{DF}	Data Float after $\overline{RD}$ ↑		50	200	50	100	ns

WRITE OPERATION

Symbol	Parameter		8231A-8 Min.	8231A-8 Max.	8231A Min.	8231A Max.	Units
t_{AW}	A_0, $\overline{CS}$ Setup to $\overline{WR}$		0		0		ns
t_{WA}	A_0, $\overline{CS}$ Hold after $\overline{WR}$		60		25		ns
t_{WY}	READY ↓ from $\overline{WR}$ ↓ Delay (Note 2)			150		100	ns
t_{YW}	READY ↑ to $\overline{WR}$ ↑		0		0		ns
t_{RRW}	READY Pulse Width (Note 4)			50		50	ns
t_{WI}	Write Inactive Time (Note 4)	Command	$4\ t_{CY}$		$4\ t_{CY}$		ns
		Data	$5\ t_{CY}$		$5\ t_{CY}$		ns
t_{DW}	Data Setup to $\overline{WR}$		150		100		ns
t_{WD}	Data Hold after $\overline{WR}$		20		20		ns

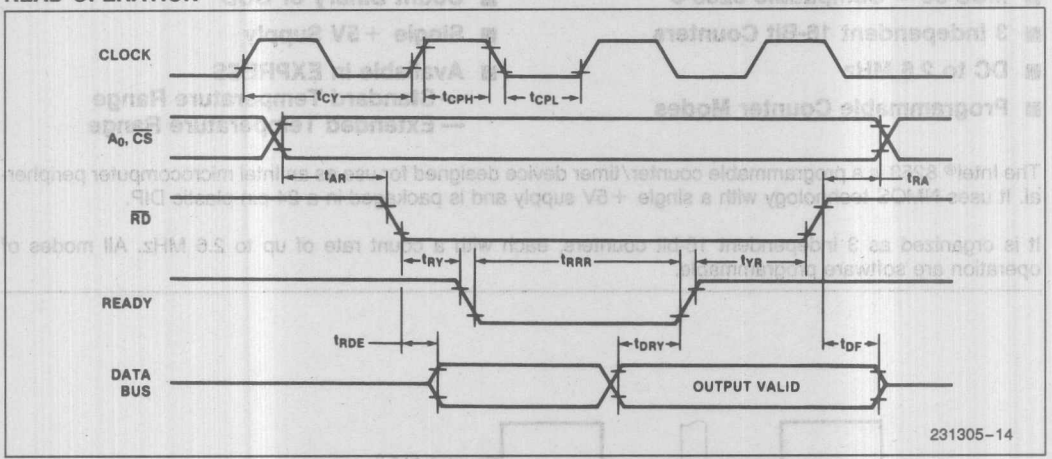
OTHER TIMINGS

Symbol	Parameter	8231A-8		8231A		Units
		Min	Max	Min	Max	
t_{CY}	Clock Period	480	5000	250	2500	ns
t_{CPH}	Clock Pulse High Width	200		100		ns
t_{CPL}	Clock Pulse Low Width	240		120		ns
t_{EE}	$\overline{END}$ Pulse Width (Note 5)	400		200		ns
t_{EAE}	$\overline{EACK}$ ↓ to $\overline{END}$ ↑ Delay		200		150	ns
t_{AA}	$\overline{EACK}$ Pulse Width	100		50		ns
t_{SA}	$\overline{SVACK}$ ↓ to SVREQ ↓ Delay		300		150	ns
t_{SS}	$\overline{SVACK}$ Pulse Width	100		50		ns

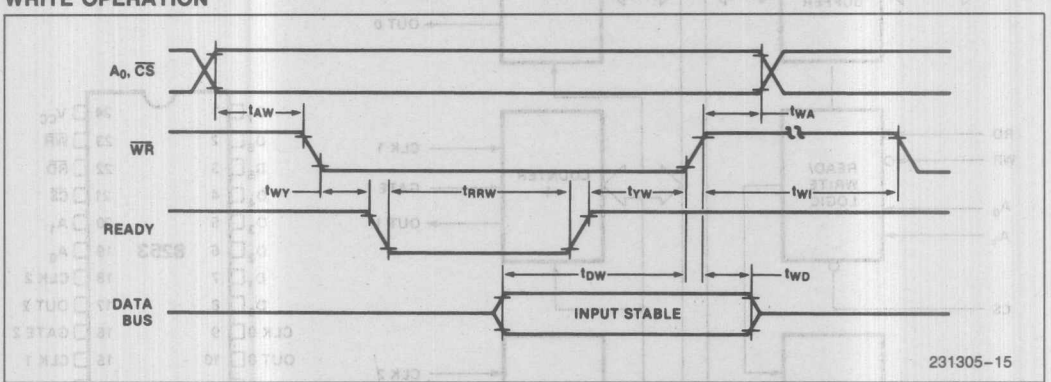
NOTES:
1. Typical values are for T_A = 25°C, nominal supply voltages processing parameters.
2. READY is pulled low for both command and data operations.
3. Minimum values shown assume no previously entered command is being executed for the data access. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. Status may be read at any time without exceeding the time shown.
4. READY low pulse width is less than 50 ns when writing into the data port or the control port as long as the duty cycle requirement (t_{WI}) is observed and no previous command is being executed. t_{WI} may be safely violated as long as the extended t_{RRW} that results is observed. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. These timings refer specifically to the 8231A.
5. $\overline{END}$ low pulse width is specified for $\overline{EACK}$ tied to VSS. Otherwise t_{EAE} applies.

WAVEFORMS

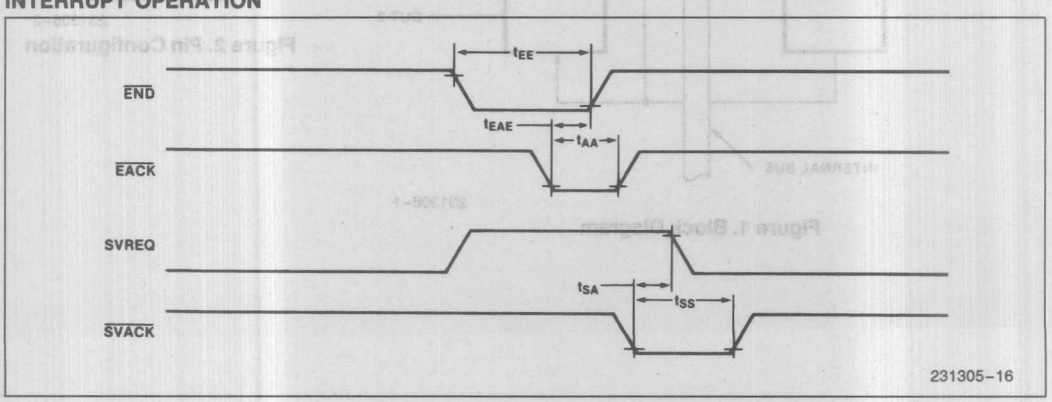
READ OPERATION



WRITE OPERATION



INTERRUPT OPERATION

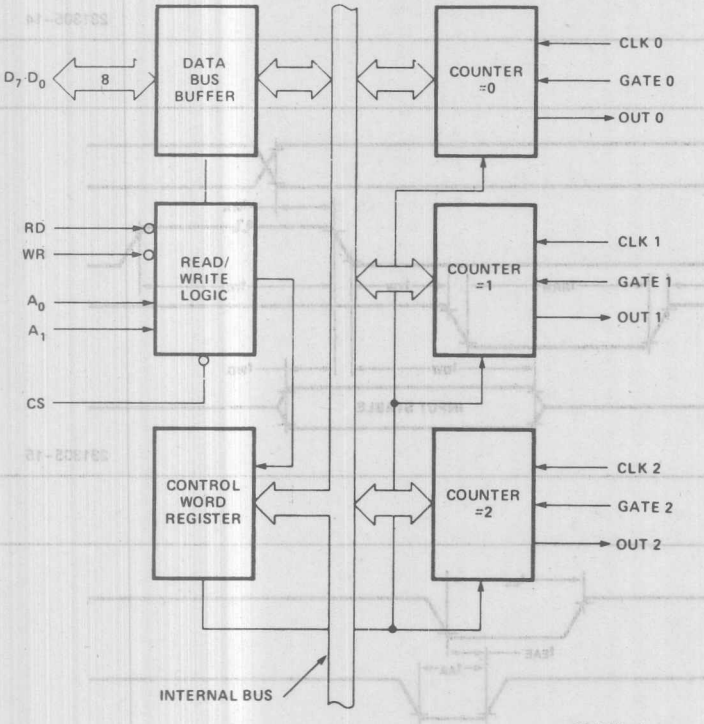


8253/8253-5
PROGRAMMABLE INTERVAL TIMER

- MCS-85™ Compatible 8253-5
- 3 Independent 16-Bit Counters
- DC to 2.6 MHz
- Programmable Counter Modes
- Count Binary or BCD
- Single +5V Supply
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

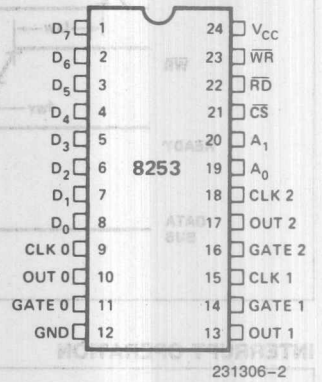
The Intel® 8253 is a programmable counter/timer device designed for use as an Intel microcomputer peripheral. It uses NMOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

It is organized as 3 independent 16-bit counters, each with a count rate of up to 2.6 MHz. All modes of operation are software programmable.



231306-1

Figure 1. Block Diagram



231306-2

Figure 2. Pin Configuration

November 1986
Order Number: 231306-001

FUNCTIONAL DESCRIPTION

General

The 8253 is programmable interval timer/counter specifically designed for use with the Intel™ Microcomputer systems. Its function is that of a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8253 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in systems software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its tasks. It is easy to see that the software overhead is minimal and that multiple delays can easily be maintained by assignment of priority levels.

Other counter/timer functions that are non-delay in nature but also common to most microcomputers can be implemented with the 8253.

- Programmable Rate Generator
- Event Counter
- Binary Rate Multiplier
- Real Time Clock
- Digital One-Shot
- Complex Motor Controller

Data Bus Buffer

The 3-state, bi-directional, 8-bit buffer is used to interface the 8253 to the system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput CPU instructions. The Data Bus Buffer has three basic functions.

1. Programming the MODES of the 8253.
2. Loading the count registers.
3. Reading the count values.

Read/Write Logic

The Read/Write Logic accepts inputs from the system bus and in turn generates control signals for overall device operation. It is enabled or disabled by CS so that no operation can occur to change the function unless the device has been selected by the system logic.

$\overline{RD}$ (Read)

A "low" on this input informs the 8253 that the CPU is inputting data in the form of a counters value.

$\overline{WR}$ (Write)

A "low" on this input informs the 8253 that the CPU is outputting data in the form of mode information or loading counters.

A0, A1

These inputs are normally connected to the address bus. Their function is to select one of the three counters to be operated on and to address the control word register for mode selection.

$\overline{CS}$ (Chip Select)

A "low" on this input enables the 8253. No reading or writing will occur unless the device is selected. The $\overline{CS}$ input has no effect upon the actual operation of the counters.

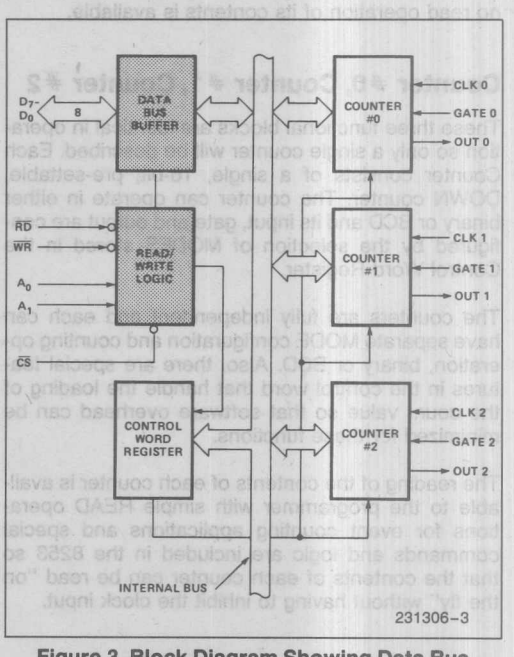


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Load Counter No. 0
0	1	0	0	1	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	0	1	1	Write Mode Word
0	0	1	0	0	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	0	1	1	0	Read Counter No. 2
0	0	1	1	1	No-Operation 3-State
1	X	X	X	X	Disable 3-State
0	1	1	X	X	No-Operation 3-State

Control Word Register

The Control Word Register is selected when A0, A1 are 11. It then accepts information from the data bus buffer and stores it in a register. The information stored in this register controls the operation MODE of each counter, selection of binary or BCD counting and the loading of each count register.

The Control Word Register can only be written into; no read operation of its contents is available.

Counter #0, Counter #1, Counter #2

These three functional blocks are identical in operation so only a single counter will be described. Each Counter consists of a single, 16-bit, pre-settable, DOWN counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the Control Word Register.

The counters are fully independent and each can have separate MODE configuration and counting operation, binary or BCD. Also, there are special features in the control word that handle the loading of the count value so that software overhead can be minimized for these functions.

The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications and special commands and logic are included in the 8253 so that the contents of each counter can be read "on the fly" without having to inhibit the clock input.

8253 SYSTEM INTERFACE

The 8253 is a component of the Intel™ Microcomputer systems and interfaces in the same manner as all other peripherals of the family. It is treated by the system software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A0, A1 connect to the A0, A1 address bus signals of the CPU. The $\overline{CS}$ can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.

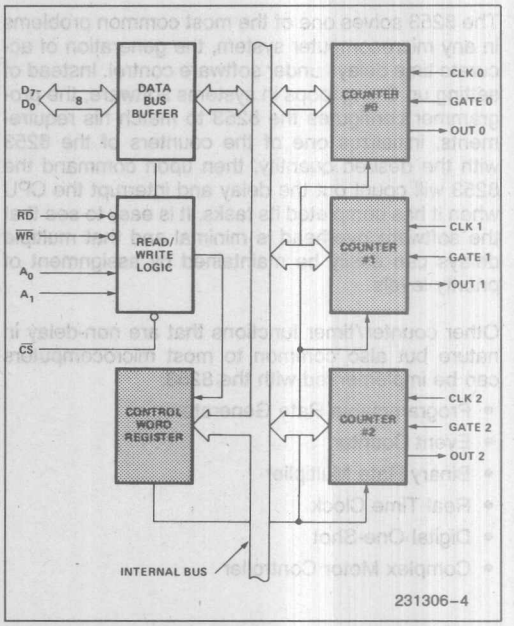


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

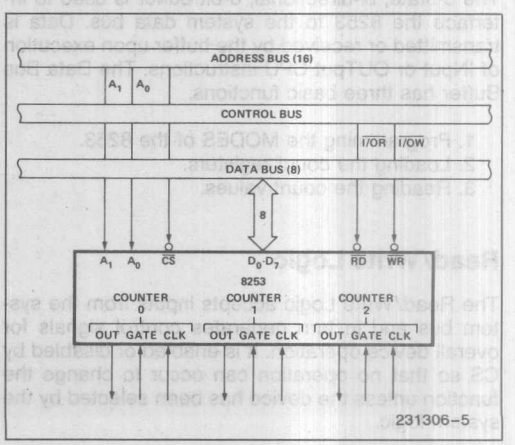


Figure 5. 8253 System Interface

OPERATIONAL DESCRIPTION

General

The complete functional definition of the 8253 is programmed by the systems software. A set of control words *must* be sent out by the CPU to initialize each counter of the 8253 with the desired MODE and quantity information. Prior to initialization, the MODE, count, and output of all counters is undefined. These control words program the MODE, Loading sequence and selection of binary or BCD counting.

Once programmed, the 8253 is ready to perform whatever timing tasks it is assigned to accomplish.

The actual counting operation of each counter is completely independent and additional logic is provided on-chip so that the usual problems associated with efficient monitoring and management of external, asynchronous events or rates to the microcomputer system have been eliminated.

Programming the 8253

All of the MODES for each counter are programmed by the systems software by simple I/O operations.

Each counter of the 8253 is individually programmed by writing a control word into the Control Word Register. (A0, A1 = 11)

Control Word Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

Definition Of Control

SC—SELECT COUNTER:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL—READ/LOAD:

RL1	RL0	
0	0	Counter Latching operation (see READ/WRITE Procedure Section).
1	0	Read/Load most significant byte only.
0	1	Read/Load least significant byte only.
1	1	Read/Load least significant byte first, then most significant byte.

M—MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-Bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Counter Loading

The count register is not loaded until the count value is written (one or two bytes, depending on the mode selected by the RL bits), followed by a rising edge and a falling edge of the clock. Any read of the counter prior to that falling clock edge may yield invalid data.

MODE DEFINITION

MODE 0: Interrupt on Terminal Count. The output will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low and the counter will count. When terminal count is reached, the output will go high and remain high until the selected count register is reloaded with the mode or a new count is loaded. The counter continues to decrement after terminal count has been reached.

Rewriting a counter register during counting results in the following:

(1) Write 1st byte stops the current counting.
(2) Write 2nd byte starts the new count.

MODE 1: Programmable One-Shot. The output will go low on the count following the rising edge of the gate input.

The output will go high on the terminal count. If a new count value is loaded while the output is low it will not affect the duration of the one-shot pulse until the succeeding trigger. The current count can be read at any time without affecting the one-shot pulse.

The one-shot is retriggerable, hence the output will remain low for the full count after any rising edge of the gate input.

MODE 2: Rate Generator. Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected, but the subsequent period will reflect the new value.

The gate input, when low, will force the output high. When the gate input goes high, the counter will start from the initial count. Thus, the gate input can be used to synchronize the counter.

When this mode is set, the output will remain high until after the count register is loaded. The output then can also be synchronized by software.

MODE 3: Square Wave Rate Generator. Similar to MODE 2 except that the output will remain high until one half the count has been completed (or even numbers) and go low for the other half of the count. This is accomplished by decrementing the counter by two on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.

If the count is odd and the output is high, the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the clock by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the counter by 3. Subsequent clock pulses decrement the count by 2 until timeout. Then the whole process is repeated. In this way, if the count is odd, the output will be high for $(N + 1)/2$ counts and low for $(N - 1)/2$ counts.

In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

MODE 4: Software Triggered Strobe. After the mode is set, the output will be high. When the count is loaded, the counter will begin counting. On terminal count, the output will go low for one input clock period, then will go high again.

If the count register is reloaded during counting, the new count will be loaded on the next CLK pulse. The count will be inhibited while the GATE input is low.

MODE 5: Hardware Triggered Strobe. The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will not go low until the full count after the rising edge of any trigger.

Signal Status Modes	Low Or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets output after next clock	—
2	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

Figure 6. Gate Pin Operations Summary

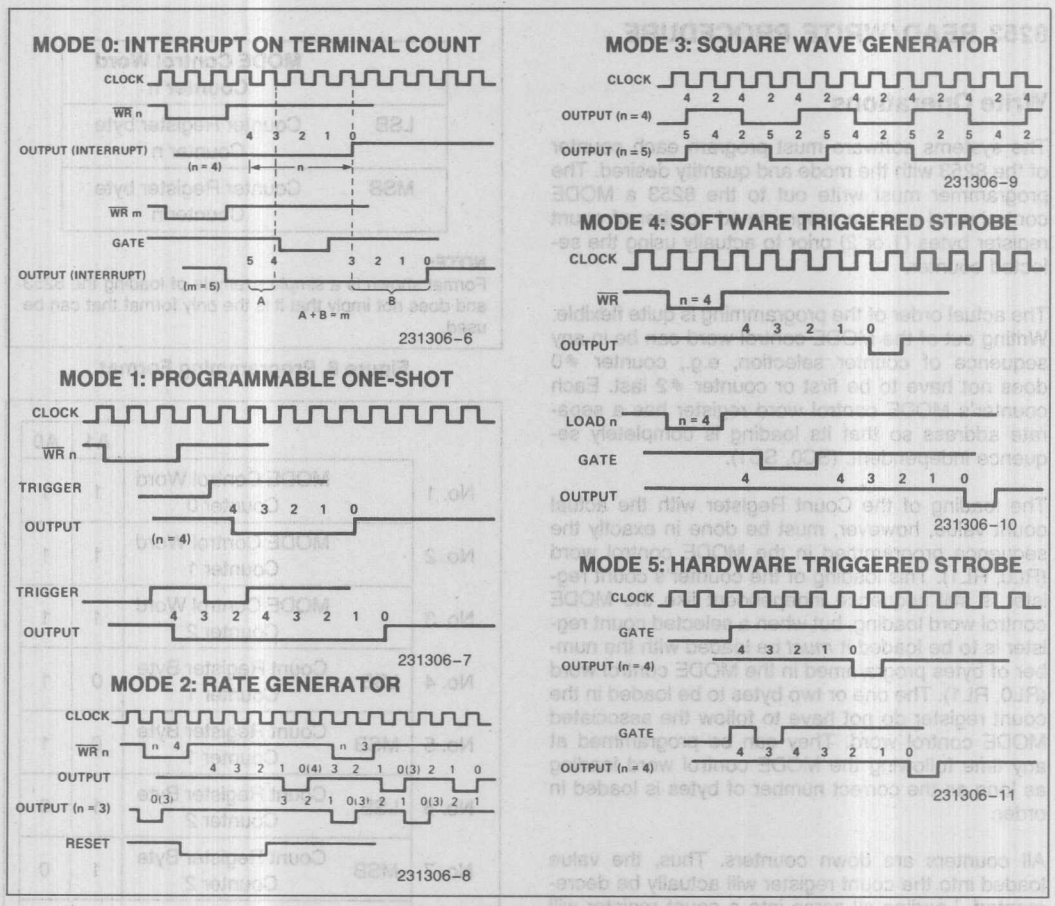


Figure 7. 8253 Timing Diagrams

8253 READ/WRITE PROCEDURE

Write Operations

The systems software must program each counter of the 8253 with the mode and quantity desired. The programmer must write out to the 8253 a MODE control word and the programmed number of count register bytes (1 or 2) prior to actually using the selected counter.

The actual order of the programming is quite flexible. Writing out of the MODE control word can be in any sequence of counter selection, e.g., counter #0 does not have to be first or counter #2 last. Each counter's MODE control word register has a separate address so that its loading is completely sequence independent. (SC0, SC1).

The loading of the Count Register with the actual count value, however, must be done in exactly the sequence programmed in the MODE control word (RL0, RL1). This loading of the counter's count register is still sequence independent like the MODE control word loading, but when a selected count register is to be loaded it *must* be loaded with the number of bytes programmed in the MODE control word (RL0, RL1). The one or two bytes to be loaded in the count register do not have to follow the associated MODE control word. They can be programmed at any time following the MODE control word loading as long as the correct number of bytes is loaded in order.

All counters are down counters. Thus, the value loaded into the count register will actually be decremented. Loading all zeros into a count register will result in the maximum count (2^{16} for Binary or 10^4 for BCD). In MODE 0 the new count will not restart until the load has been completed. It will accept one of two bytes depending on how the MODE control words (RL0, RL1) are programmed. Then proceed with the restart operation.

	MODE Control Word Counter n
LSB	Counter Register byte Counter n
MSB	Counter Register byte Counter n

NOTE:
Format shown is a simple example of loading the 8253 and does not imply that it is the only format that can be used.

Figure 8. Programming Format

			A1	**A0**
No. 1		MODE Control Word Counter 0	1	1
No. 2		MODE Control Word Counter 1	1	1
No. 3		MODE Control Word Counter 2	1	1
No. 4	LSB	Count Register Byte Counter 1	0	1
No. 5	MSB	Count Register Byte Counter 1	0	1
No. 6	LSB	Count Register Byte Counter 2	1	0
No. 7	MSB	Count Register Byte Counter 2	1	0
No. 8	LSB	Count Register Byte Counter 0	0	0
No. 9	MSB	Count Register Byte Counter 0	0	0

NOTE:
The exclusive addresses of each counter's count register make the task of programming the 8253 a very simple matter, and maximum effective use of the device will result if this feature is fully initilized.

Figure 9. Alternate Programming Formats

Read Operations

In most counter applications it becomes necessary to read the value of the count in progress and make a computational decision based on this quantity. Event counters are probably the most common application that uses this function. The 8253 contains logic that will allow the programmer to easily read the contents of any of the three counters without disturbing the actual count in progress.

There are two methods that the programmer can use to read the value of the counters. The first method involves the use of simple I/O read operations of the selected counter. By controlling the A0, A1 inputs to the 8253 the programmer can select the counter to be read (remember that no read operation of the mode register is allowed A0, A1-11). The only requirement with this method is that in order to assure a stable count reading the actual operation of the selected counter *must be inhibited* either by controlling the Gate input or by external logic that inhibits the clock input. The contents of the counter selected will be available as follows:

First I/O Read contains the least significant byte (LSB).

Second I/O Read contains the most significant byte (MSB).

Due to the internal logic of the 8253 it is absolutely necessary to complete the entire reading procedure. If two bytes are programmed to be read, then two bytes *must* be read before any loading WR command can be sent to the same counter.

Read Operation Chart

A1	A0	RD	
0	0	0	Read Counter No. 0
0	1	0	Read Counter No. 1
1	0	0	Read Counter No. 2
1	1	0	Illegal

Reading While Counting

In order for the programmer to read the contents of any counter without effecting or disturbing the counting operation the 8253 has special internal logic that can be accessed using simple $\overline{WR}$ commands to the MODE register. Basically, when the programmer wishes to read the contents of a selected counter "on the fly" he loads the MODE register with a special code which latches the present count value into a storage register so that its contents contain an accurate, stable quantity. The programmer then issues a normal read command to the selected counter and the contents of the latched register is available.

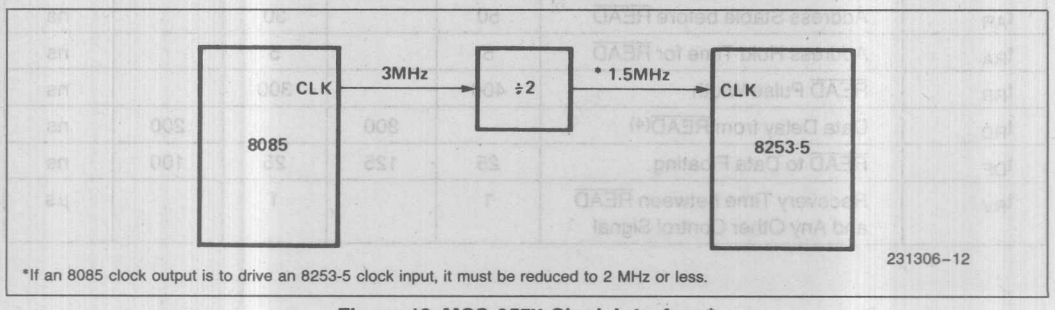
MODE Register for Latching Count

A0, A1 = 11

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	X	X	X	X

SC1, SC0— specify counter to be latched.
D5, D4 — 00 designates counter latching operation.
X — don't care.

The same limitation applies to this mode of reading the counter as the previous method. That is, it is mandatory to complete the entire read operation as programmed. This command has no effect on the counter's mode.



*If an 8085 clock output is to drive an 8253-5 clock input, it must be reduced to 2 MHz or less.

Figure 10. MCS-85™ Clock Interface*

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage On Any Pin
with Respect to Ground . . . . . −0.5V to 7V
Power Dissipation . . . . . 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%*

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.2	V_{CC} + .5V	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		140	mA	

CAPACITANCE T_A = 25°C, V_{CC} = GND = 0V

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V*

Bus Parameters(3)

READ CYCLE

Symbol	Parameter	8253		8253-5		Unit
		Min	Max	Min	Max	
t_{AR}	Address Stable before $\overline{READ}$	50		30		ns
t_{RA}	Address Hold Time for $\overline{READ}$	5		5		ns
t_{RR}	$\overline{READ}$ Pulse Width	400		300		ns
t_{RD}	Data Delay from $\overline{READ}$(4)		300		200	ns
t_{DF}	$\overline{READ}$ to Data Floating	25	125	25	100	ns
t_{RV}	Recovery Time between $\overline{READ}$ and Any Other Control Signal	1		1		μs

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8253		8253-5		Unit
		Min	Max	Min	Max	
t_{AW}	Address Stable before $\overline{WRITE}$	50		30		ns
t_{WA}	Address Hold Time for $\overline{WRITE}$	30		30		ns
t_{WW}	$\overline{WRITE}$ Pulse Width	400		300		ns
t_{DW}	Data Set Up Time for $\overline{WRITE}$	300		250		ns
t_{WD}	Data Hold Time for $\overline{WRITE}$	40		30		ns
t_{RV}	Recovery Time between $\overline{WRITE}$ and Any Other Control Signal	1		1		μs

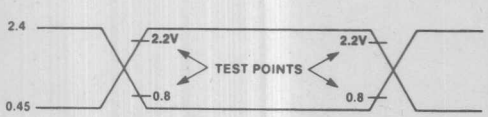
CLOCK AND GATE TIMING

Symbol	Parameter	8253		8253-5		Unit
		Min	Max	Min	Max	
t_{CLK}	Clock Period	380	dc	380	dc	ns
t_{PWH}	High Pulse Width	230		230		ns
t_{PWL}	Low Pulse Width	150		150		ns
t_{GW}	Gate Width High	150		150		ns
t_{GL}	Gate Width Low	100		100		ns
t_{GS}	Gate Set Up Time to CLK ↑	100		100		ns
t_{GH}	Gate Hold Time after CLK ↑	50		50		ns
t_{OD}	Output Delay from CLK ↓ (4)		400		400	ns
t_{ODG}	Output Delay from Gate ↓ (4)		300		300	ns

NOTES:
1. I_{OL} = 2.2 mA.
2. I_{OH} = −400 μA.
3. AC timings measured at V_{OH} 2.2, V_{OL} = 0.8.
4. C_L = 150 pF.

*For Extended Temperature EXPRESS, use M8253 electrical parameters.

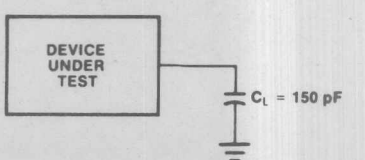
A.C. TESTING INPUT, OUTPUT WAVEFORM



231306-13

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.2V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT

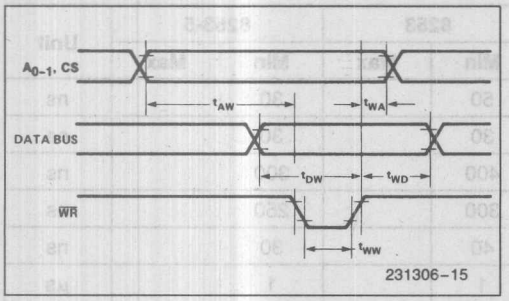


231306-14

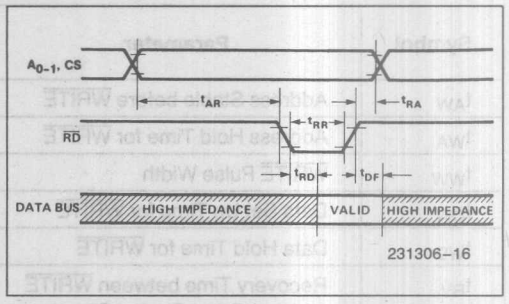
C_L Includes Jig Capacitance

WAVEFORMS

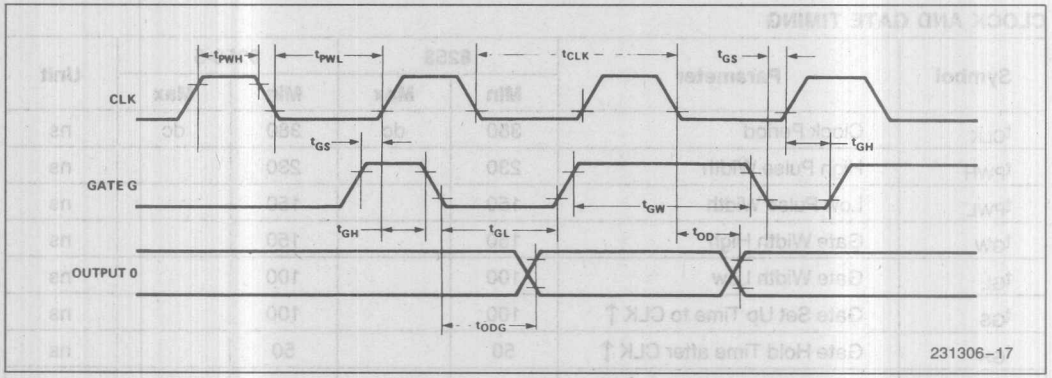
WRITE TIMING



READ TIMING



CLOCK AND GATE TIMING



8254
PROGRAMMABLE INTERVAL TIMER

- **Compatible with All Intel and Most Other Microprocessors**
- **Handles Inputs from DC to 10 MHz**
  - — 5 MHz 8254-5
  - — 8 MHz 8254
  - — 10 MHz 8254-2
- **Status Read-Back Command**
- **Six Programmable Counter Modes**
- **Three Independent 16-Bit Counters**
- **Binary or BCD Counting**
- **Single +5V Supply**
- **Available in EXPRESS**
  - — Standard Temperature Range

The Intel® 8254 is a counter/timer device designed to solve the common timing control problems in microcomputer system design. It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz. All modes are software programmable. The 8254 is a superset of the 8253.

The 8254 uses HMOS technology and comes in a 24-pin plastic or CERDIP package.

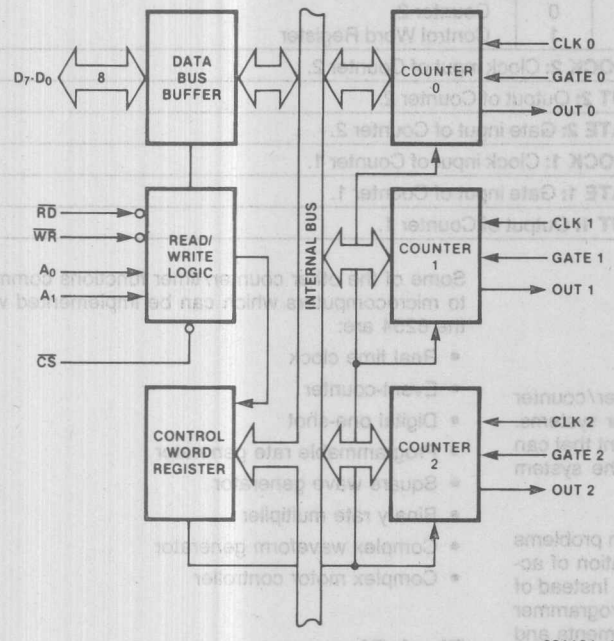


Figure 1. 8254 Block Diagram

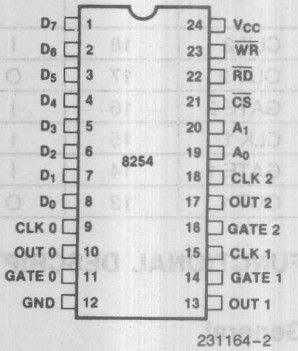


Figure 2. Pin Configuration

August 1987
Order Number: 231164-004

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
D_7–D_0	1–8	I/O	**DATA:** Bi-directional three state data bus lines, connected to system data bus.
CLK 0	9	I	**CLOCK 0:** Clock input of Counter 0.
OUT 0	10	O	**OUTPUT 0:** Output of Counter 0.
GATE 0	11	I	**GATE 0:** Gate input of Counter 0.
GND	12		**GROUND:** Power supply connection.
V_{CC}	24		**POWER:** +5V power supply connection.
$\overline{WR}$	23	I	**WRITE CONTROL:** This input is low during CPU write operations.
$\overline{RD}$	22	I	**READ CONTROL:** This input is low during CPU read operations.
$\overline{CS}$	21	I	**CHIP SELECT:** A low on this input enables the 8254 to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and $\overline{WR}$ are ignored otherwise.
A_1, A_0	20–19	I	**ADDRESS:** Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.

A_1	A_0	Selects
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word Register

CLK 2	18	I	**CLOCK 2:** Clock input of Counter 2.
OUT 2	17	O	**OUT 2:** Output of Counter 2.
GATE 2	16	I	**GATE 2:** Gate input of Counter 2.
CLK 1	15	I	**CLOCK 1:** Clock input of Counter 1.
GATE 1	14	I	**GATE 1:** Gate input of Counter 1.
OUT 1	13	O	**OUT 1:** Output of Counter 1.

FUNCTIONAL DESCRIPTION

General

The 8254 is a programmable interval timer/counter designed for use with Intel microcomputer systems. It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8254 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 8254 to match his requirements and programs one of the counters for the desired delay. After the desired delay, the 8254 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 8254 are:

- Real time clock
- Event-counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

Block Diagram

DATA BUS BUFFER

This 3-state, bi-directional, 8-bit buffer is used to interface the 8254 to the system bus (see Figure 3).

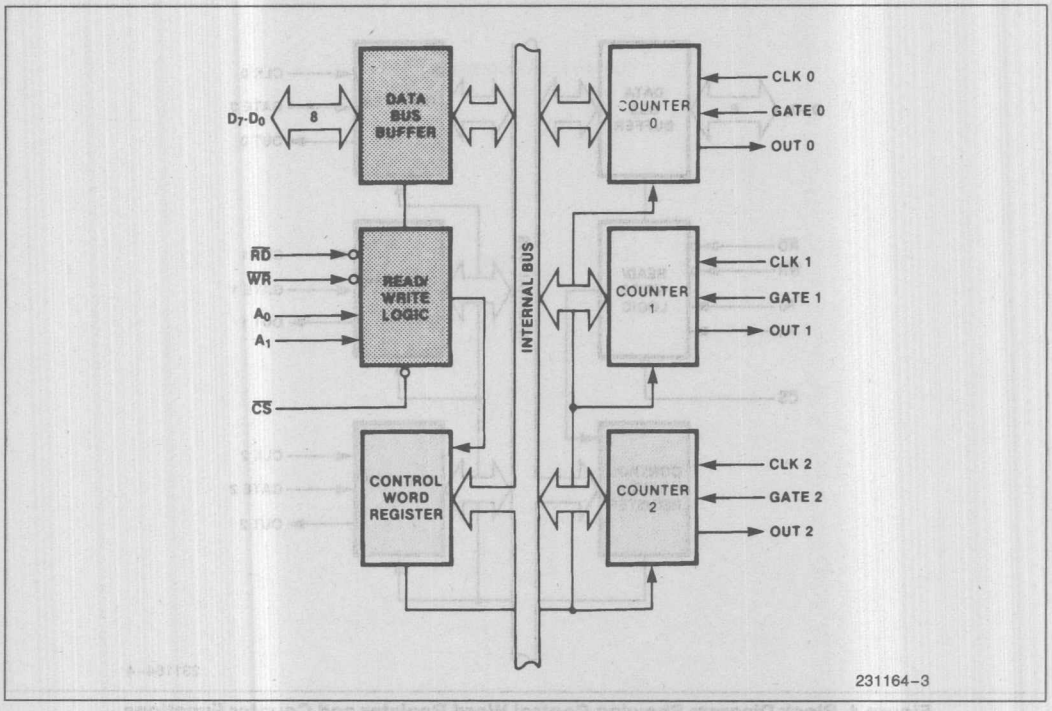


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

READ/WRITE LOGIC

The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254. A_1 and A_0 select one of the three counters or the Control Word Register to be read from/written into. A "low" on the $\overline{RD}$ input tells the 8254 that the CPU is reading one of the counters. A "low" on the $\overline{WR}$ input tells the 8254 that the CPU is writing either a Control Word or an initial count. Both $\overline{RD}$ and $\overline{WR}$ are qualified by $\overline{CS}$; $\overline{RD}$ and $\overline{WR}$ are ignored unless the 8254 has been selected by holding $\overline{CS}$ low.

CONTROL WORD REGISTER

The Control Word Register (see Figure 4) is selected by the Read/Write Logic when $A_1,A_0 = 11$. If the CPU then does a write operation to the 8254, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

The Control Word Register can only be written to; status information is available with the Read-Back Command.

COUNTER 0, COUNTER 1, COUNTER 2

These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.

The Counters are fully independent. Each Counter may operate in a different Mode.

The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.

The status register, shown in Figure 5, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)

The actual counter is labelled CE (for "Counting Element"). It is a 16-bit presettable synchronous down counter.

OL_M and OL_L are two 8-bit latches. OL stands for "Output Latch"; the subscripts M and L stand for "Most significant byte" and "Least significant byte"

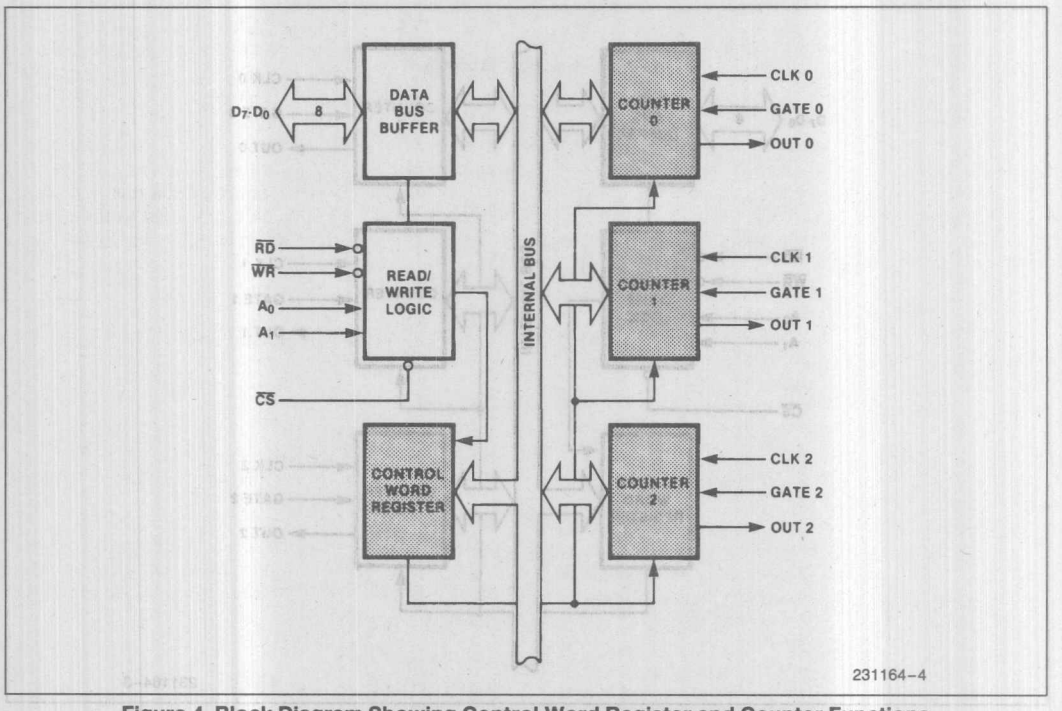


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

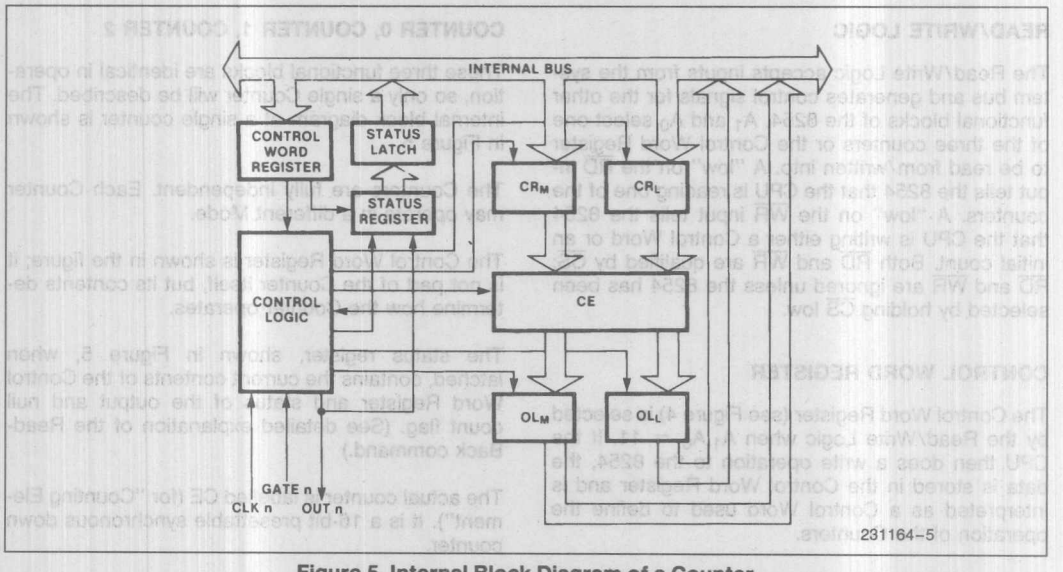


Figure 5. Internal Block Diagram of a Counter

respectively. Both are normally referred to as one unit and called just OL. These latches normally "follow" the CE, but if a suitable Counter Latch Command is sent to the 8254, the latches "latch" the present count until read by the CPU and then return to "following" the CE. One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

Similarly, there are two 8-bit registers called CR_M and CR_L (for "Count Register"). Both are normally referred to as one unit and called just CR. When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously. CR_M and CR_L are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero. Note that the CE cannot be written into; whenever a count is written, it is written into the CR.

The Control Logic is also shown in the diagram. CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.

8254 SYSTEM INTERFACE

The 8254 is a component of the Intel Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the system's software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A_0,A_1 connect to the A_0, A_1 address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.

OPERATIONAL DESCRIPTION

General

After power-up, the state of the 8254 is undefined. The Mode, count value, and output of all Counters are undefined.

How each Counter operates is determined when it is programmed. Each Counter must be programmed before it can be used. Unused counters need not be programmed.

Programming the 8254

Counters are programmed by writing a Control Word and then an initial count.

The Control Words are written into the Control Word Register, which is selected when A_1,A_0 = 11. The Control Word itself specifies which Counter is being programmed.

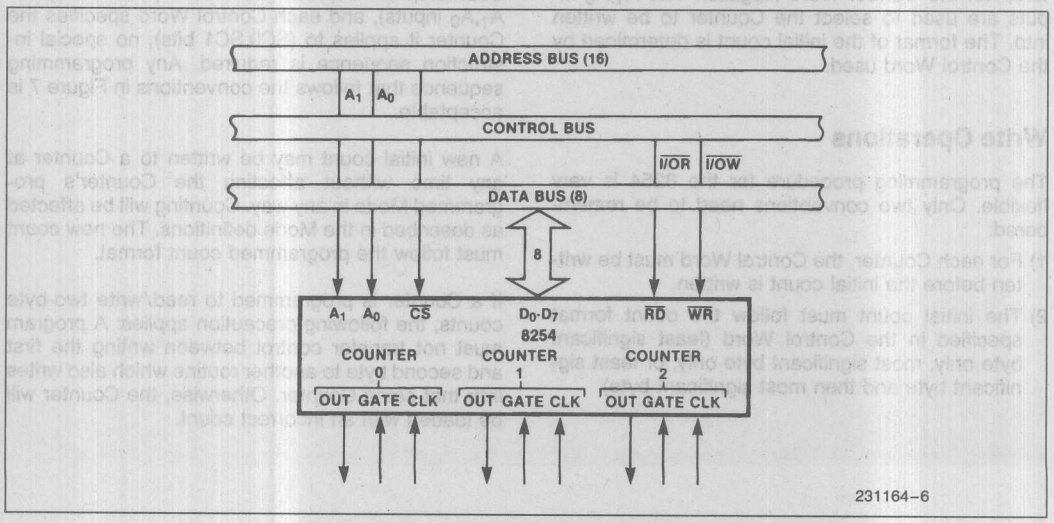


Figure 6. 8254 System Interface

Control Word Format

$A_1,A_0 = 11$ $\overline{CS} = 0$ $\overline{RD} = 1$ $\overline{WR} = 0$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 7. Control Word Format

By contrast, initial counts are written into the Counters, not the Control Word Register. The A_1,A_0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.

Since the Control Word Register and the three Counters have separate addresses (selected by the A_1,A_0 inputs), and each Control Word specifies the Counter it applies to (SC0,SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions in Figure 7 is acceptable.

Write Operations

The programming procedure for the 8254 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.
2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

	A_1	A_0
Control Word—Counter 0	1	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0
Control Word—Counter 1	1	1
LSB of count—Counter 1	0	1
MSB of count—Counter 1	0	1
Control Word—Counter 2	1	1
LSB of count—Counter 2	1	0
MSB of count—Counter 2	1	0

	A_1	A_0
Control Word—Counter 2	1	1
Control Word—Counter 1	1	1
Control Word—Counter 0	1	1
LSB of count—Counter 2	1	0
MSB of count—Counter 2	1	0
LSB of count—Counter 1	0	1
MSB of count—Counter 1	0	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0

	A_1	A_0
Control Word—Counter 0	1	1
Control Word—Counter 1	1	1
Control Word—Counter 2	1	1
LSB of count—Counter 2	1	0
LSB of count—Counter 1	0	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0
MSB of count—Counter 1	0	1
MSB of count—Counter 2	1	0

	A_1	A_0
Control Word—Counter 1	1	1
Control Word—Counter 0	1	1
LSB of count—Counter 1	0	1
Control Word—Counter 2	1	1
LSB of count—Counter 0	0	0
MSB of count—Counter 1	0	1
LSB of count—Counter 2	1	0
MSB of count—Counter 0	0	0
MSB of count—Counter 2	1	0

NOTE:
In all four examples, all Counters are programmed to read/write two-byte counts. These are only four of many possible programming sequences.

Figure 8. A Few Possible Programming Sequences

Read Operations

It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 8254.

There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command. Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

COUNTER LATCH COMMAND

The second method uses the "Counter Latch Command". Like a Control Word, this command is written to the Control Word Register, which is selected when $A_1,A_0 = 11$. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

$A_1,A_0 = 11$; CS = 0; RD = 1; WR = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1,SC0—specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4—00 designates Counter Latch Command

X—don't care

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 9. Counter Latching Command Format

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to "following" the counting element (CE). This allows reading the contents of the Counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other Counters may be inserted between them.

Another feature of the 8254 is that reads and writes of the same Counter may be interleaved; for example, if the Counter is programmed for two byte counts, the following sequence is valid.

1) Read least significant byte.
2) Write new least significant byte.
3) Read most significant byte.
4) Write new most significant byte.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

READ-BACK COMMAND

The third method uses the Read-Back Command. This command allows the user to check the count value, programmed Mode, and current states of the OUT pin and Null Count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3, D2, D1 = 1.

A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{COUNT}$	$\overline{STATUS}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counters(s)
D_3: 1 = Select Counter 2
D_2: 1 = Select Counter 1
D_1: 1 = Select Counter 0
D_0: Reserved for future expansion; Must be 0

Figure 10. Read-Back Command Format

The read-back command may be used to latch multiple counter output latches (OL) by setting the $\overline{COUNT}$ bit D5 = 0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). The counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

The read-back command may also be used to latch status information of selected counter(s) by setting $\overline{STATUS}$ bit D4 = 0. Status must be latched to be read; status of a counter is accessed by a read from that counter.

The counter status format is shown in Figure 11. Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Output	Null Count	RW1	RW0	M2	M1	M0	BCD

D_7 1 = OUT Pin is 1
0 = OUT Pin is 0

D_6 1 = Null Count
0 = Count available for reading

D_5–D_0 Counter programmed mode (see Figure 7)

Figure 11. Status Byte

NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

This Action	Causes
A. Write to the control word register;(1)	Null Count = 1
B. Write to the count register (CR);(2)	Null Count = 1
C. New Count is loaded into CE (CR → CE);	Null Count = 0

NOTE:
1. Only the counter specified by the control word will have its Null Count set to 1. Null count bits of other counters are unaffected.
2. If the counter is programmed for two-byte counts (least significant byte then most significant byte) Null Count goes to 1 when the second byte is written.

Figure 12. Null Count Operation

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both $\overline{COUNT}$ and $\overline{STATUS}$ bits D5,D4 = 0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.

If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Write into Counter 0
0	1	0	0	1	Write into Counter 1
0	1	0	1	0	Write into Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read from Counter 0
0	0	1	0	1	Read from Counter 1
0	0	1	1	0	Read from Counter 2
0	0	1	1	1	No-Operation (3-State)
1	X	X	X	X	No-Operation (3-State)
0	1	1	X	X	No-Operation (3-State)

Figure 14. Read/Write Operations Summary

Command								Description	Result
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	1	0	0	0	0	1	0	Read back count and status of Counter 0	Count and status latched for Counter 0
1	1	1	0	0	1	0	0	Read back status of Counter 1	Status latched for Counter 1
1	1	1	0	1	1	0	0	Read back status of Counters 2, 1	Status latched for Counter 2, but not Counter 1
1	1	0	1	1	0	0	0	Read back count of Counter 2	Count latched for Counter 2
1	1	0	0	0	1	0	0	Read back count and status of Counter 1	Count latched for Counter 1, but not status
1	1	1	0	0	0	1	0	Read back status of Counter 1	Command ignored, status already latched for Counter 1

Figure 13. Read-Back Command Example

Mode Definitions

The following are defined for use in describing the operation of the 8254.

CLK Pulse: a rising edge, then a falling edge, in that order, of a Counter's CLK input.

Trigger: a rising edge of a Counter's GATE input.

Counter loading: the transfer of a count from the CR to the CE (refer to the "Functional Description")

MODE 0: INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N + 1 CLK pulses after the initial count is written.

If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required)
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N+1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.

MODE 1: HARDWARE RETRIGGERABLE ONE-SHOT

OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero. OUT will then go high and remain high until the CLK pulse after the next trigger.

After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration. The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected unless the counter is retriggered. In that case, the Counter is loaded with the new count and the one-shot pulse continues until the new count expires.

MODE 2: RATE GENERATOR

This Mode functions like a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt. OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated. Mode 2 is periodic; the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

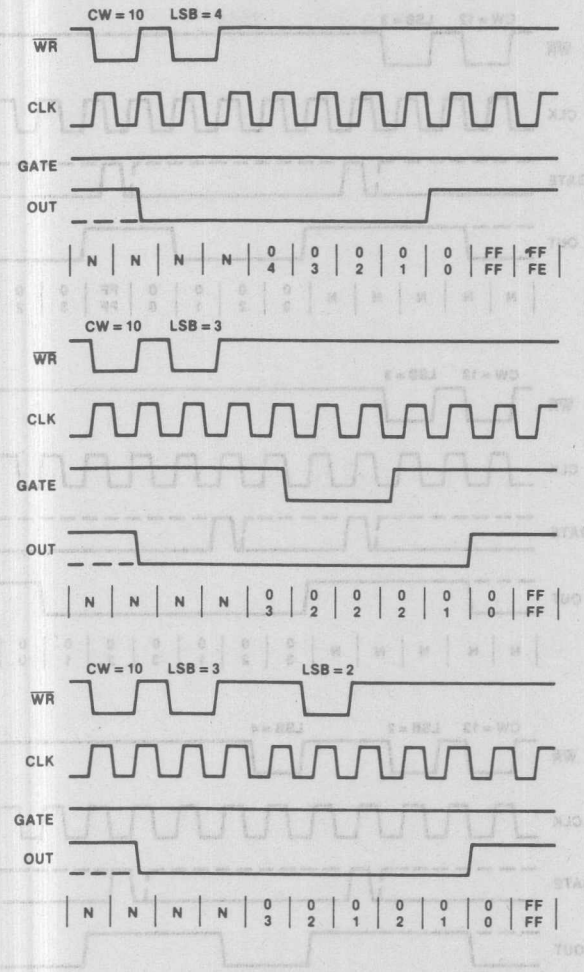
GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately. A trigger reloads the Counter with the initial count on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.

MODE 3: SQUARE WAVE MODE

Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the



NOTE:
The following conventions apply to all mode timing diagrams:
1. Counters are programmed for binary (not BCD) counting and for reading/writing least significant byte (LSB) only.
2. The counter is always selected ($\overline{CS}$ always low).
3. CW stands for "Control Word"; CW = 10 means a control word of 10 HEX is written to the counter.
4. LSB stands for "Least Significant Byte" of count.
5. Numbers below diagrams are count values. The lower number is the least significant byte. The upper number is the most significant byte. Since the counter is programmed to read/write LSB only, the most significant byte cannot be read.
   N stands for an undefined count.
   Vertical lines show transitions between count values.

Figure 15. Mode 0

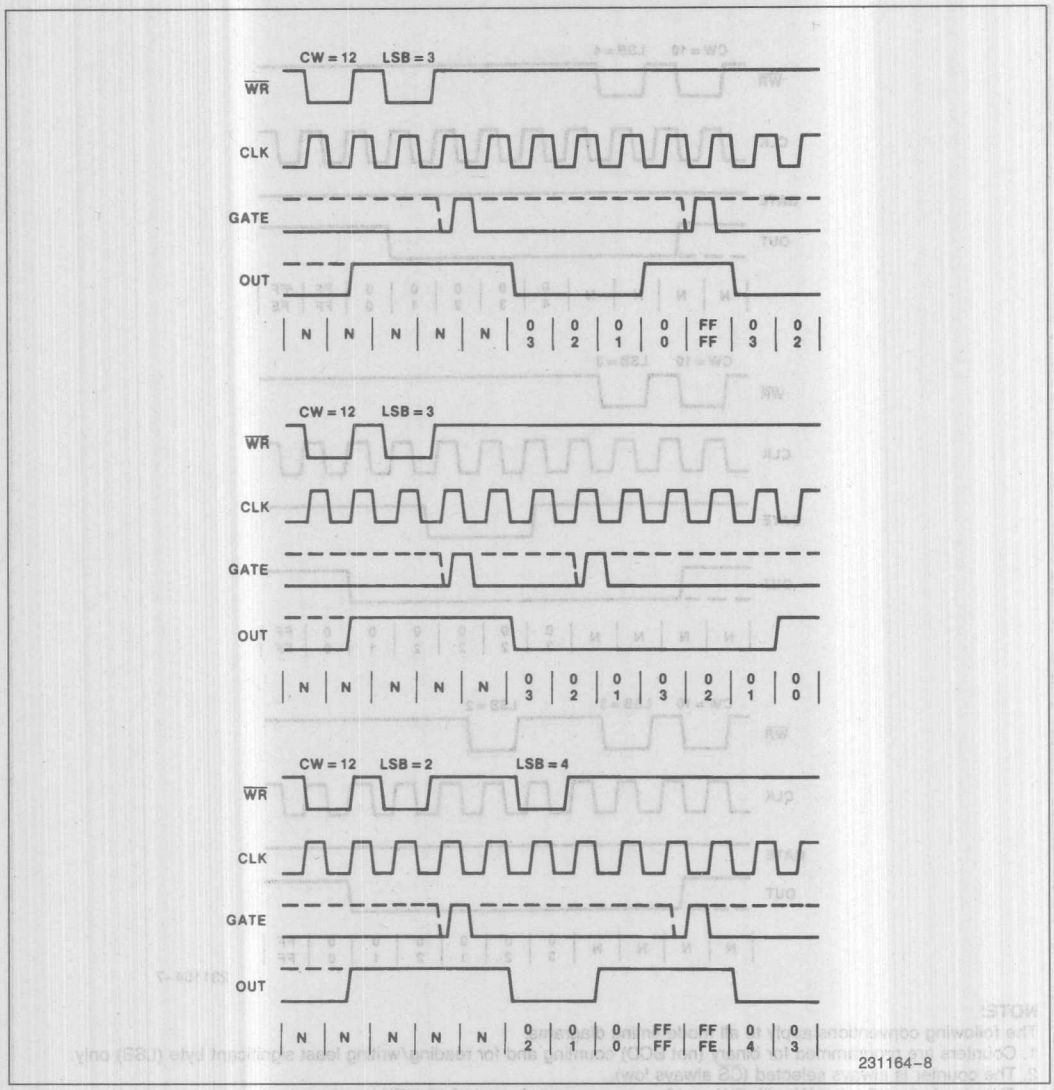


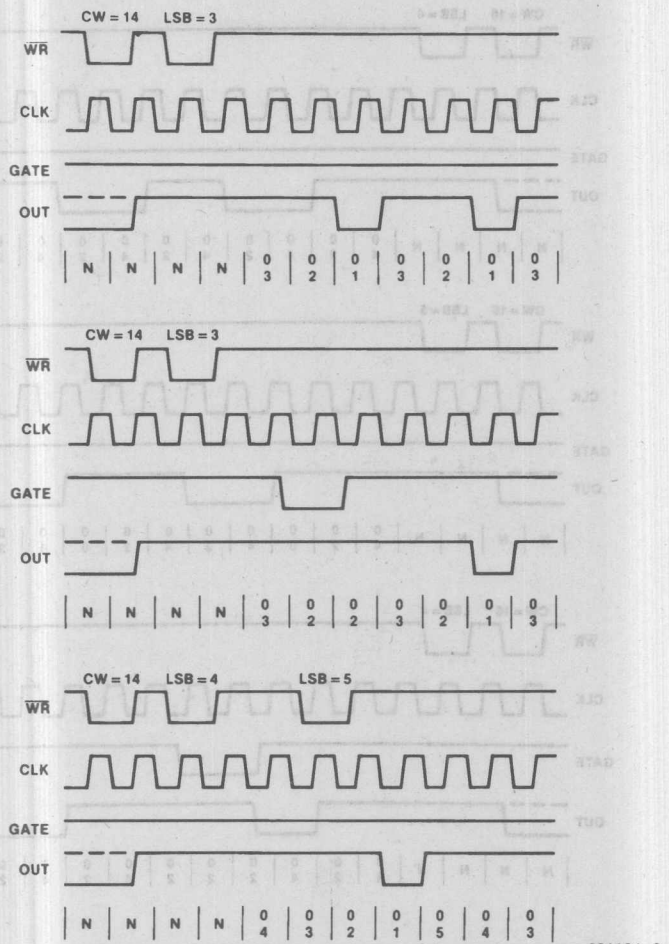
Figure 16. Mode 1

initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required. A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the



NOTE:
A GATE transition should not occur one clock prior to terminal count.

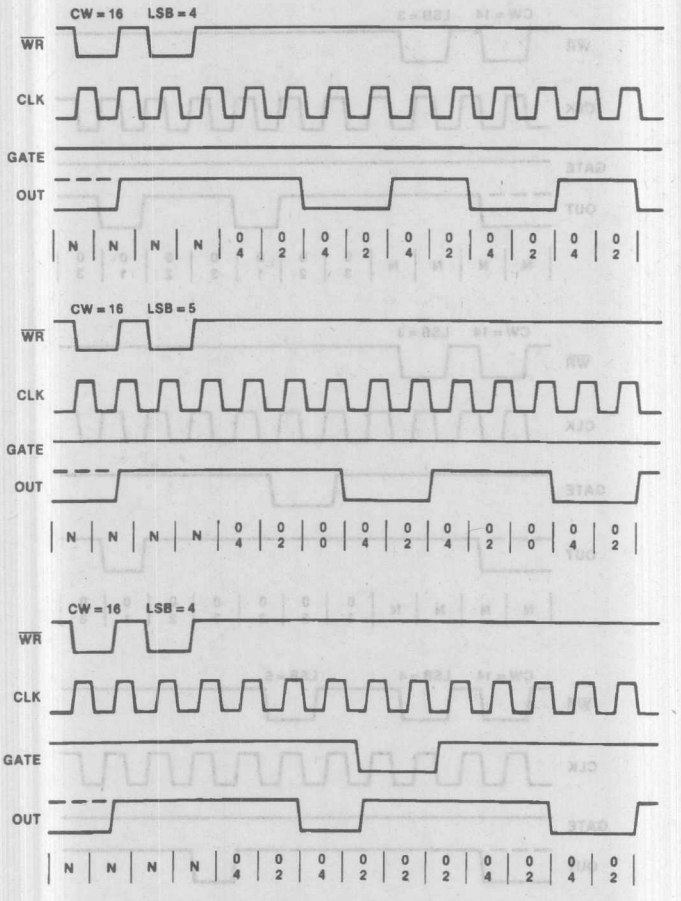
Figure 17. Mode 2

new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

Mode 3 is implemented as follows:

Even counts: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.

Odd counts: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. One CLK pulse *after* the count expires, OUT goes low and the Counter is reloaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT will be high for $(N + 1)/2$ counts and low for $(N - 1)/2$ counts.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 18. Mode 3

MODE 4: SOFTWARE TRIGGERED STROBE

OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is "triggered" by writing the initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be "retriggered" by software. OUT strobes low N + 1 CLK pulses after the new count of N is written.

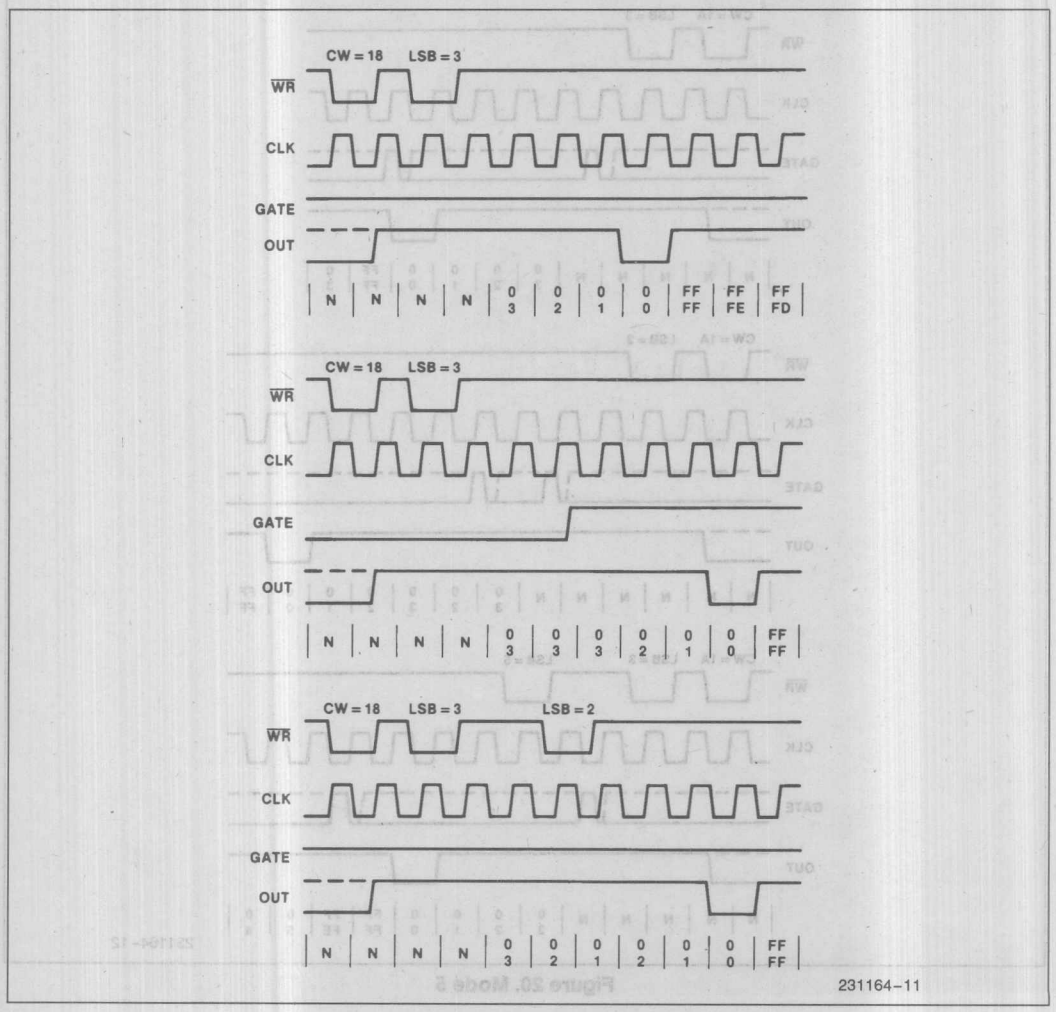


Figure 19. Mode 4

MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after a trigger.

A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N + 1 CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

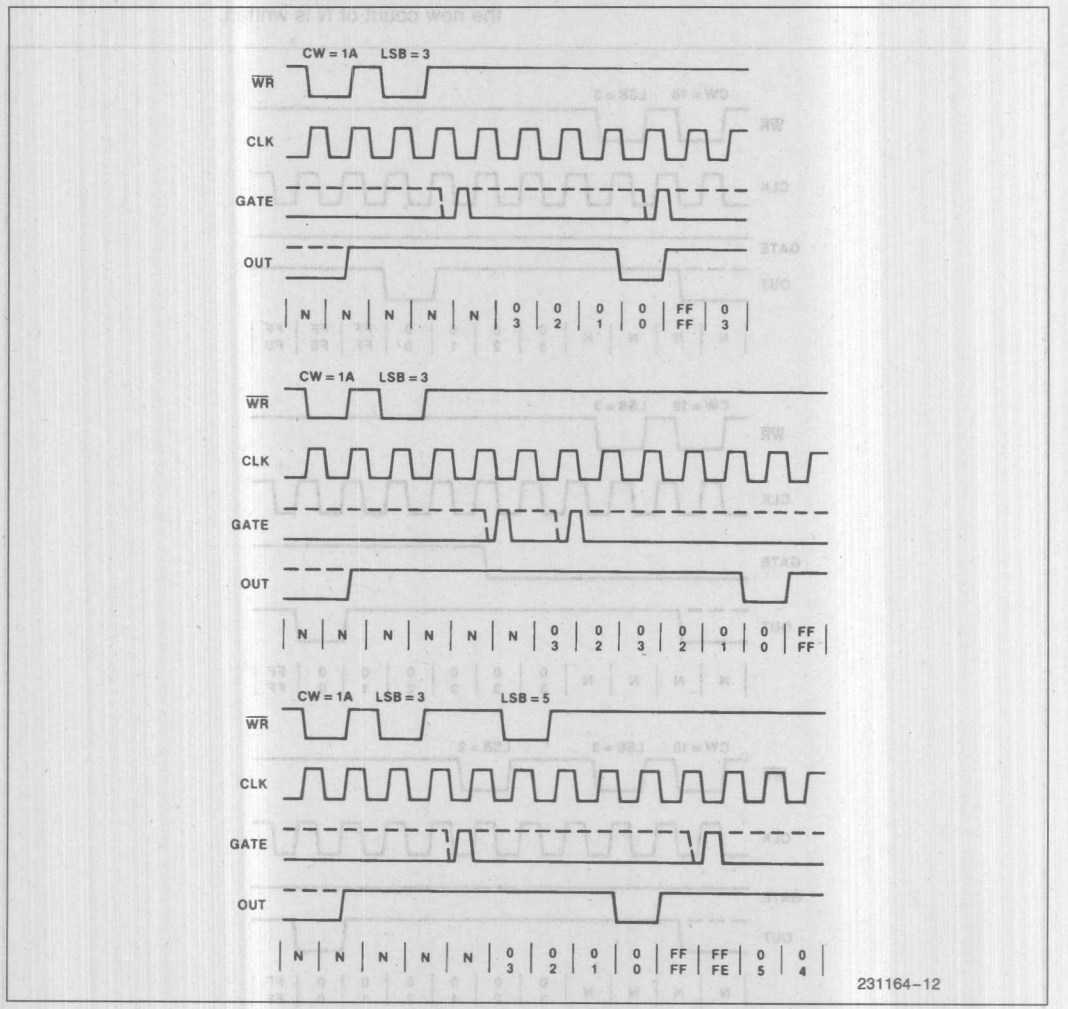


Figure 20. Mode 5

Signal Status Modes	Low Or Going Low	Rising	High
0	Disables Counting	— —	Enables Counting
1	— —	1) Initiates Counting 2) Resets Output after Next Clock	— —
2	1) Disables Counting 2) Sets Output Immediately High	Initiates Counting	Enables Counting
3	1) Disables Counting 2) Sets Output Immediately High	Initiates Counting	Enables Counting
4	Disables Counting	— —	Enables Counting
5	— —	Initiates Counting	— —

Figure 21. Gate Pin Operations Summary

Mode	Min Count	Max Count
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

NOTE:
0 is equivalent to 2^{16} for binary counting and 10^4 for BCD counting.

Figure 22. Minimum and Maximum Initial Counts

Operation Common to All Modes

PROGRAMMING

When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.

GATE

The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive. In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs—a high logic level does not have to be maintained until the next rising edge of CLK. Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

COUNTER

New counts are loaded and Counters are decremented on the falling edge of CLK.

The largest possible initial count is 0; this is equivalent to 2^{16} for binary counting and 10^4 for BCD counting.

The Counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5 the Counter "wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground.....−0.5V to +7V
Power Dissipation 1W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		170	mA	
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}(4)

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%, GND = 0V

Bus Parameters(1)

READ CYCLE

Symbol	Parameter	8254-5		8254		8254-2		Unit
		Min	Max	Min	Max	Min	Max	
t_{AR}	Address Stable Before $\overline{RD}$ ↓	45		45		30		ns
t_{SR}	$\overline{CS}$ Stable Before $\overline{RD}$ ↓	0		0		0		ns
t_{RA}	Address Hold Time After $\overline{RD}$ ↑	0		0		0		ns
t_{RR}	$\overline{RD}$ Pulse Width	150		150		95		ns
t_{RD}	Data Delay from $\overline{RD}$ ↓		120		120		85	ns
t_{AD}	Data Delay from Address		220		220		185	ns
t_{DF}	$\overline{RD}$ ↑ to Data Floating	5	90	5	90	5	65	ns
t_{RV}	Command Recovery Time	200		200		165		ns

NOTE:
1. AC timings measured at V_{OH} = 2.0V, V_{OL} = 0.8V.

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%, GND = 0V (Continued)

WRITE CYCLE

Symbol	Parameter	8254-5		8254		8254-2		Unit
		Min	Max	Min	Max	Min	Max	
t_{AW}	Address Stable Before $\overline{WR}$ ↓	0		0		0		ns
t_{SW}	$\overline{CS}$ Stable Before $\overline{WR}$ ↓	0		0		0		ns
$t_{\overline{WA}}$	Address Hold Time After $\overline{WR}$ ↓	0		0		0		ns
t_{WW}	$\overline{WR}$ Pulse Width	150		150		95		ns
t_{DW}	Data Setup Time Before $\overline{WR}$ ↑	120		120		95		ns
t_{WD}	Data Hold Time After $\overline{WR}$ ↑	0		0		0		ns
t_{RV}	Command Recovery Time	200		200		165		ns

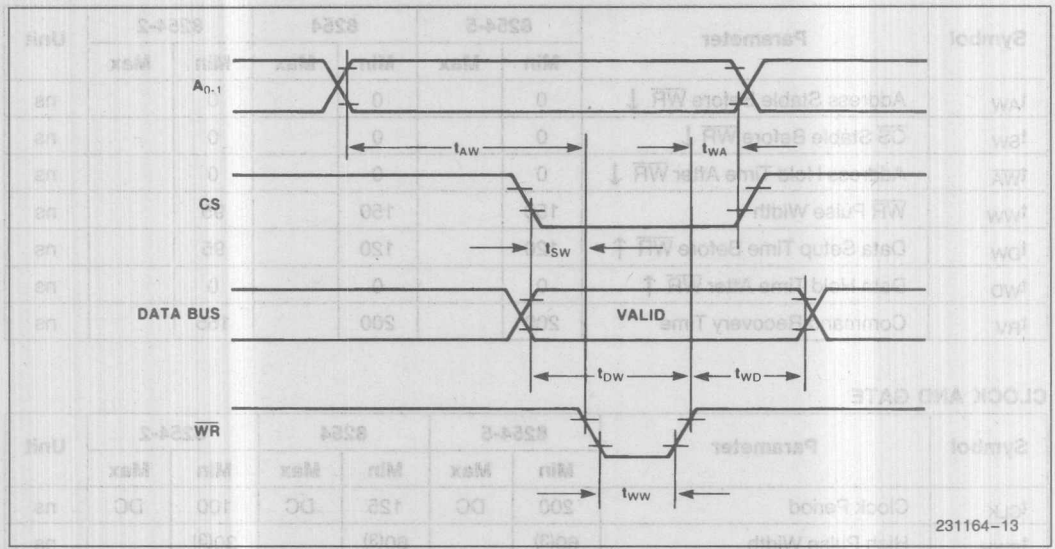
CLOCK AND GATE

Symbol	Parameter	8254-5		8254		8254-2		Unit
		Min	Max	Min	Max	Min	Max	
t_{CLK}	Clock Period	200	DC	125	DC	100	DC	ns
t_{PWH}	High Pulse Width	60(3)		60(3)		30(3)		ns
t_{PWL}	Low Pulse Width	60(3)		60(3)		50(3)		ns
t_R	Clock Rise Time		25		25		25	ns
t_F	Clock Fall Time		25		25		25	ns
t_{GW}	Gate Width High	50		50		50		ns
t_{GL}	Gate Width Low	50		50		50		ns
t_{GS}	Gate Setup Time to CLK ↑	50		50		40		ns
t_{GH}	Gate Setup Time After CLK ↑	50(2)		50(2)		50(2)		ns
t_{OD}	Output Delay from CLK ↓		150		150		100	ns
t_{ODG}	Output Delay from Gate ↓		120		120		100	ns
t_{WC}	CLK Delay for Loading ↓	0	55	0	55	0	55	ns
t_{WG}	Gate Delay for Sampling	−5	50	−5	50	−5	40	ns
t_{WO}	OUT Delay from Mode Write		260		260		240	ns
t_{CL}	CLK Set Up for Count Latch	−40	45	−40	45	−40	40	ns

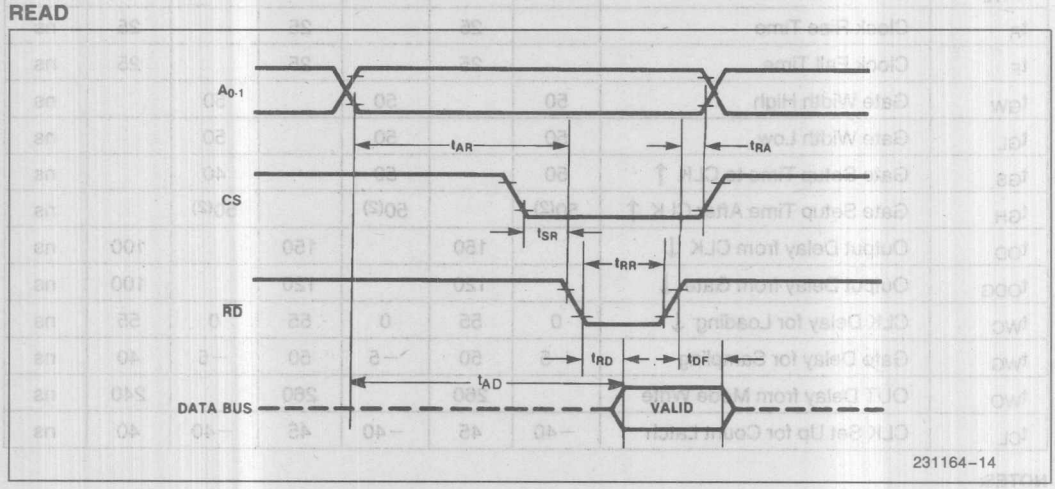
NOTES:
2. In Modes 1 and 5 triggers are sampled on each rising clock edge. A second trigger within 120 ns (70 ns for the 8254-2) of the rising clock edge may not be detected.
3. Low-going glitches that violate t_{PWH}, t_{PWL} may cause errors requiring counter reprogramming.
4. Sampled, not 100% tested. T_A = 25°C.
5. If CLK present at TWC min then Count equals N+2 CLK pulses, TWC max equals Count N+1 CLK pulse. TWC min to TWC max, count will be either N+1 or N+2 CLK pulses.
6. In Modes 1 and 5, if GATE is present when writing a new Count value, at TWG min Counter will not be triggered, at TWG max Counter will be triggered.
7. If CLK present when writing a Counter Latch or ReadBack Command, at TCL min CLK will be reflected in count value latched, at TCL max CLK will not be reflected in the count value latched.

WAVEFORMS

WRITE

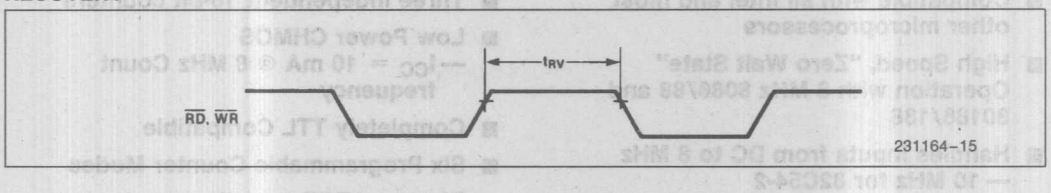


READ

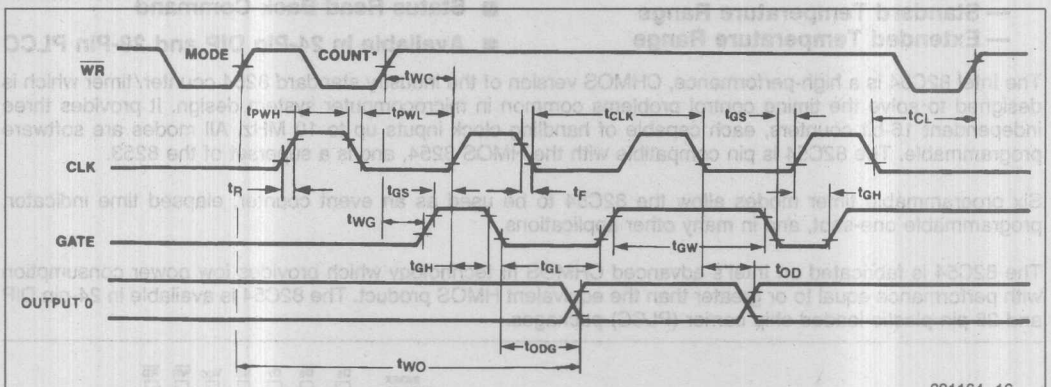


WAVEFORMS (Continued)

RECOVERY

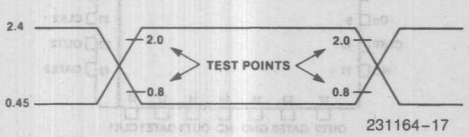


CLOCK AND GATE



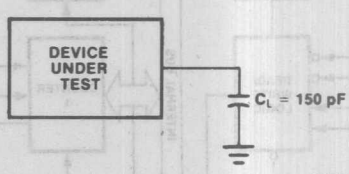
*Last byte of count being written.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0." Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L Includes Jig Capacitance

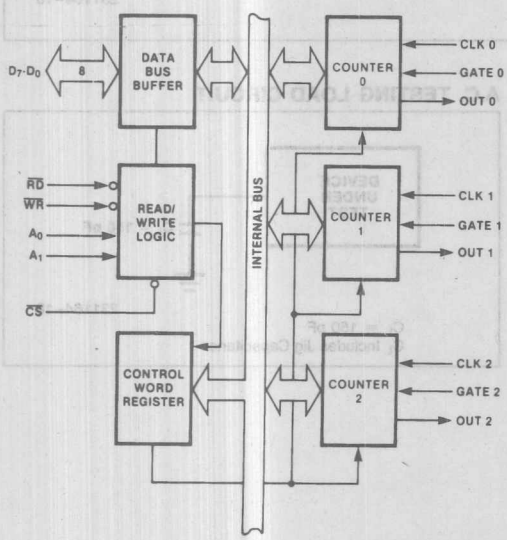
82C54
CHMOS PROGRAMMABLE INTERVAL TIMER

- **Compatible with all Intel and most other microprocessors**
- **High Speed, "Zero Wait State" Operation with 8 MHz 8086/88 and 80186/188**
- **Handles Inputs from DC to 8 MHz — 10 MHz for 82C54-2**
- **Available in EXPRESS — Standard Temperature Range — Extended Temperature Range**
- **Three independent 16-bit counters**
- **Low Power CHMOS — I_{CC} = 10 mA @ 8 MHz Count frequency**
- **Completely TTL Compatible**
- **Six Programmable Counter Modes**
- **Binary or BCD counting**
- **Status Read Back Command**
- **Available in 24-Pin DIP and 28-Pin PLCC**

The Intel 82C54 is a high-performance, CHMOS version of the industry standard 8254 counter/timer which is designed to solve the timing control problems common in microcomputer system design. It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz. All modes are software programmable. The 82C54 is pin compatible with the HMOS 8254, and is a superset of the 8253.

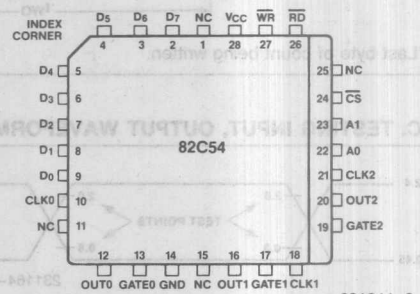
Six programmable timer modes allow the 82C54 to be used as an event counter, elapsed time indicator, programmable one-shot, and in many other applications.

The 82C54 is fabricated on Intel's advanced CHMOS III technology which provides low power consumption with performance equal to or greater than the equivalent HMOS product. The 82C54 is available in 24-pin DIP and 28-pin plastic leaded chip carrier (PLCC) packages.



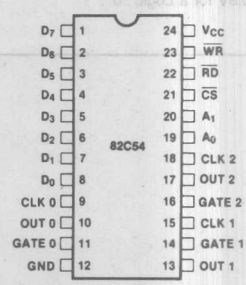
231244-1

Figure 1. 82C54 Block Diagram



231244-3

PLASTIC LEADED CHIP CARRIER



231244-2

Diagrams are for pin reference only.
Package sizes are not to scale.

Figure 2. 82C54 Pinout

October 1987
Order Number: 231244-004

Table 1. Pin Description

Symbol	Pin Number DIP	Pin Number PLCC	Type	Function
D_7-D_0	1-8	2-9	I/O	Data: Bidirectional tri-state data bus lines, connected to system data bus.
CLK 0	9	10	I	Clock 0: Clock input of Counter 0.
OUT 0	10	12	O	Output 0: Output of Counter 0.
GATE 0	11	13	I	Gate 0: Gate input of Counter 0.
GND	12	14		Ground: Power supply connection.
OUT 1	13	16	O	Out 1: Output of Counter 1.
GATE 1	14	17	I	Gate 1: Gate input of Counter 1.
CLK 1	15	18	I	Clock 1: Clock input of Counter 1.
GATE 2	16	19	I	Gate 2: Gate input of Counter 2.
OUT 2	17	20	O	Out 2: Output of Counter 2.
CLK 2	18	21	I	Clock 2: Clock input of Counter 2.
A_1, A_0	20-19	23-22	I	Address: Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.
				A_1 | A_0 | Selects 0 | 0 | Counter 0 0 | 1 | Counter 1 1 | 0 | Counter 2 1 | 1 | Control Word Register
$\overline{CS}$	21	24	I	Chip Select: A low on this input enables the 82C54 to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and $\overline{WR}$ are ignored otherwise.
$\overline{RD}$	22	26	I	Read Control: This input is low during CPU read operations.
$\overline{WR}$	23	27	I	Write Control: This input is low during CPU write operations.
V_{CC}	24	28		Power: +5V power supply connection.
NC		1, 11, 15, 25		No Connect

FUNCTIONAL DESCRIPTION

General

The 82C54 is a programmable interval timer/counter designed for use with Intel microcomputer systems. It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 82C54 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 82C54 to match his requirements and programs one of the counters for the desired delay. After the desired delay, the 82C54 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 82C54 are:

- Real time clock
- Even counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

Block Diagram

DATA BUS BUFFER

This 3-state, bi-directional, 8-bit buffer is used to interface the 82C54 to the system bus (see Figure 3).

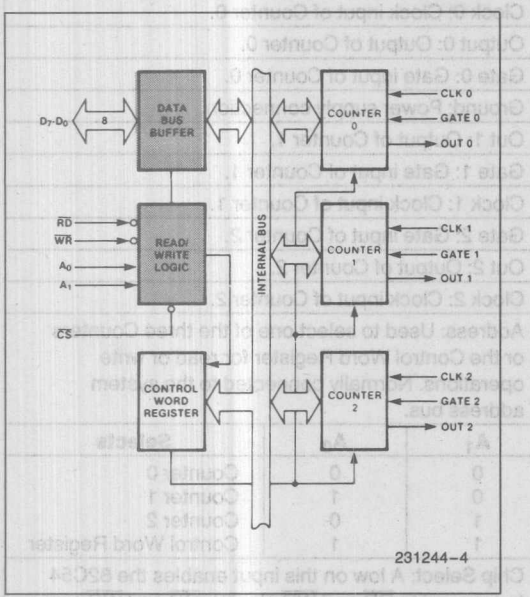


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

READ/WRITE LOGIC

The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 82C54. A_1 and A_0 select one of the three counters or the Control Word Register to be read from/written into. A "low" on the $\overline{RD}$ input tells the 82C54 that the CPU is reading one of the counters. A "low" on the $\overline{WR}$ input tells the 82C54 that the CPU is writing either a Control Word or an initial count. Both $\overline{RD}$ and $\overline{WR}$ are qualified by $\overline{CS}$; $\overline{RD}$ and $\overline{WR}$ are ignored unless the 82C54 has been selected by holding $\overline{CS}$ low.

CONTROL WORD REGISTER

The Control Word Register (see Figure 4) is selected by the Read/Write Logic when A_1, A_0 = 11. If the CPU then does a write operation to the 82C54, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

The Control Word Register can only be written to; status information is available with the Read-Back Command.

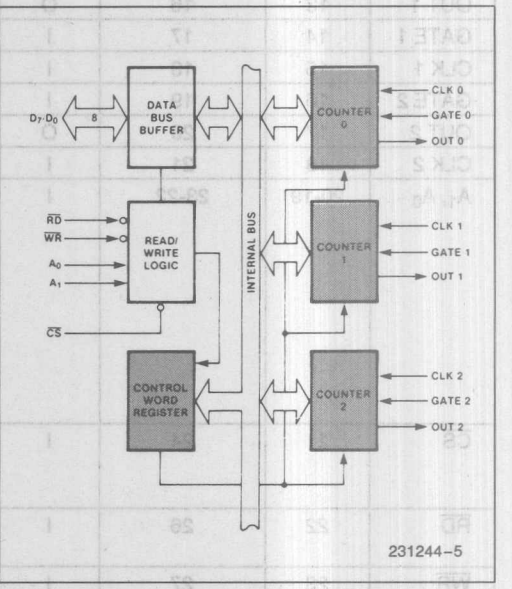


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

COUNTER 0, COUNTER 1, COUNTER 2

These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.

The Counters are fully independent. Each Counter may operate in a different Mode.

The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.

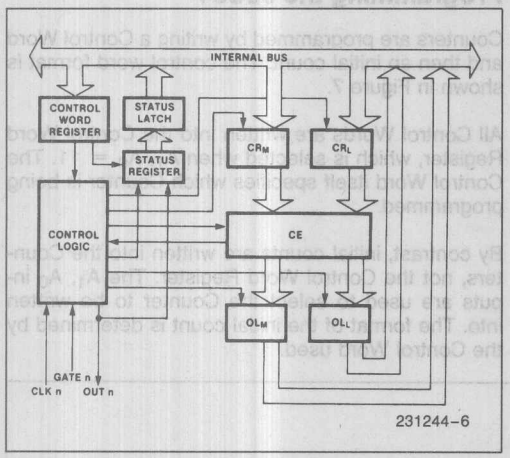


Figure 5. Internal Block Diagram of a Counter

The status register, shown in the Figure, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)

The actual counter is labelled CE (for "Counting Element"). It is a 16-bit presettable synchronous down counter.

OL_M and OL_L are two 8-bit latches. OL stands for "Output Latch"; the subscripts M and L stand for "Most significant byte" and "Least significant byte" respectively. Both are normally referred to as one unit and called just OL. These latches normally "follow" the CE, but if a suitable Counter Latch Command is sent to the 82C54, the latches "latch" the present count until read by the CPU and then return to "following" the CE. One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

Similarly, there are two 8-bit registers called CR_M and CR_L (for "Count Register"). Both are normally referred to as one unit and called just CR. When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously. CR_M and CR_L are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero. Note that the CE cannot be written into; whenever a count is written, it is written into the CR.

The Control Logic is also shown in the diagram. CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.

82C54 SYSTEM INTERFACE

The 82C54 is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A_0, A_1 connect to the A_0, A_1 address bus signals of the CPU. The $\overline{CS}$ can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.

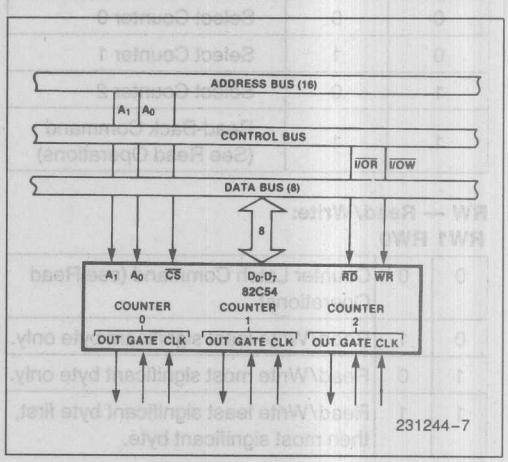


Figure 6. 82C54 System Interface

OPERATIONAL DESCRIPTION

General

After power-up, the state of the 82C54 is undefined. The Mode, count value, and output of all Counters are undefined.

How each Counter operates is determined when it is programmed. Each Counter must be programmed before it can be used. Unused counters need not be programmed.

Programming the 82C54

Counters are programmed by writing a Control Word and then an initial count. The control word format is shown in Figure 7.

All Control Words are written into the Control Word Register, which is selected when A_1, A_0 = 11. The Control Word itself specifies which Counter is being programmed.

By contrast, initial counts are written into the Counters, not the Control Word Register. The A_1, A_0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.

Control Word Format

A_1, A_0 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (See Read Operations)

RW — Read/Write:

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only.
1	0	Read/Write most significant byte only.
1	1	Read/Write least significant byte first, then most significant byte.

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Figure 7. Control Word Format

Write Operations

The programming procedure for the 82C54 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.
2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

Since the Control Word Register and the three Counters have separate addresses (selected by the A_1, A_0 inputs), and each Control Word specifies the Counter it applies to (SC0, SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

		A_1	A_0
Control Word —	Counter 0	1	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0
Control Word —	Counter 1	1	1
LSB of count —	Counter 1	0	1
MSB of count —	Counter 1	0	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 2	1	0

		A_1	A_0
Control Word —	Counter 0	1	1
Counter Word —	Counter 1	1	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 2	1	0
LSB of count —	Counter 1	0	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0
MSB of count —	Counter 1	0	1
MSB of count —	Counter 2	1	0

		A_1	A_0
Control Word —	Counter 2	1	1
Control Word —	Counter 1	1	1
Control Word —	Counter 0	1	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 2	1	0
LSB of count —	Counter 1	0	1
MSB of count —	Counter 1	0	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0

		A_1	A_0
Control Word —	Counter 1	1	1
Control Word —	Counter 0	1	1
LSB of count —	Counter 1	0	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 1	0	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 0	0	0
MSB of count —	Counter 2	1	0

NOTE:
In all four examples, all counters are programmed to read/write two-byte counts.
These are only four of many possible programming sequences.

Figure 8. A Few Possible Programming Sequences

Read Operations

It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 82C54.

There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command. Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

COUNTER LATCH COMMAND

The second method uses the "Counter Latch Command". Like a Control Word, this command is written to the Control Word Register, which is selected when A_1, A_0 = 11. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

A_1, A_0 = 11; $\overline{CS}$ = 0; $\overline{RD}$ = 1; $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1, SC0 - specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4 - 00 designates Counter Latch Command

X - don't care

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 9. Counter Latching Command Format

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to "following" the counting element (CE). This allows reading the contents of the Counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other Counters may be inserted between them.

Another feature of the 82C54 is that reads and writes of the same Counter may be interleaved; for example, if the Counter is programmed for two byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a Counter is programmed to read/write two-byte counts, the following precaution applies; A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

READ-BACK COMMAND

The third method uses the Read-Back command. This command allows the user to check the count value, programmed Mode, and current state of the OUT pin and Null Count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3,D2,D1 = 1.

A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{COUNT}$	$\overline{STATUS}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counter(s)
D_3: 1 = Select counter 2
D_2: 1 = Select counter 1
D_1: 1 = Select counter 0
D_0: Reserved for future expansion; must be 0

Figure 10. Read-Back Command Format

The read-back command may be used to latch multiple counter output latches (OL) by setting the $\overline{COUNT}$ bit D5 = 0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). That counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the

count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

The read-back command may also be used to latch status information of selected counter(s) by setting STATUS bit D4 = 0. Status must be latched to be read; status of a counter is accessed by a read from that counter.

The counter status format is shown in Figure 11. Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
OUTPUT	NULL COUNT	RW1	RW0	M2	M1	M0	BCD

D_7 1 = Out Pin is 1
0 = Out Pin is 0
D_6 1 = Null count
0 = Count available for reading
D_5-D_0 Counter Programmed Mode (See Figure 7)

Figure 11. Status Byte

NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

THIS ACTION:	CAUSES:
A. Write to the control word register:[1]	Null count = 1
B. Write to the count register (CR);[2]	Null count = 1
C. New count is loaded into CE (CR → CE);	Null count = 0

[1] Only the counter specified by the control word will have its null count set to 1. Null count bits of other counters are unaffected.

[2] If the counter is programmed for two-byte counts (least significant byte then most significant byte) null count goes to 1 when the second byte is written.

Figure 12. Null Count Operation

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both COUNT and STATUS bits D5,D4 = 0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.

If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.

Command								Description	Results
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	1	0	0	0	0	1	0	Read back count and status of Counter 0	Count and status latched for Counter 0
1	1	1	0	0	1	0	0	Read back status of Counter 1	Status latched for Counter 1
1	1	1	0	1	1	0	0	Read back status of Counters 2, 1	Status latched for Counter 2, but not Counter 1
1	1	0	1	1	0	0	0	Read back count of Counter 2	Count latched for Counter 2
1	1	0	0	0	1	0	0	Read back count and status of Counter 1	Count latched for Counter 1, but not status
1	1	1	0	0	0	1	0	Read back status of Counter 1	Command ignored, status already latched for Counter 1

Figure 13. Read-Back Command Example

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Write into Counter 0
0	1	0	0	1	Write into Counter 1
0	1	0	1	0	Write into Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read from Counter 0
0	0	1	0	1	Read from Counter 1
0	0	1	1	0	Read from Counter 2
0	0	1	1	1	No-Operation (3-State)
1	X	X	X	X	No-Operation (3-State)
0	1	1	X	X	No-Operation (3-State)

Figure 14. Read/Write Operations Summary

Mode Definitions

The following are defined for use in describing the operation of the 82C54.

CLK PULSE: a rising edge, then a falling edge, in that order, of a Counter's CLK input.

TRIGGER: a rising edge of a Counter's GATE input.

COUNTER LOADING: the transfer of a count from the CR to the CE (refer to the "Functional Description")

MODE 0: INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

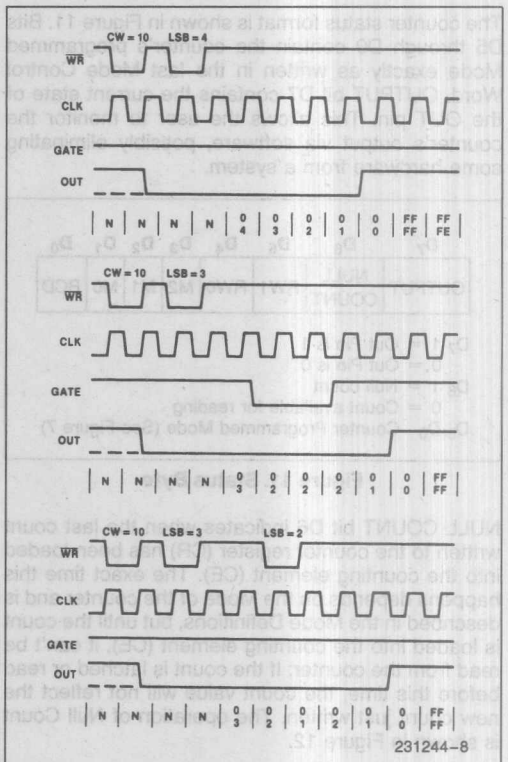
After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N + 1 CLK pulses after the initial count is written.

If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required).
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N + 1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.



NOTE:
The Following Conventions Apply To All Mode Timing Diagrams:
1. Counters are programmed for binary (not BCD) counting and for Reading/Writing least significant byte (LSB) only.
2. The counter is always selected ($\overline{CS}$ always low).
3. CW stands for "Control Word"; CW = 10 means a control word of 10, hex is written to the counter.
4. LSB stands for "Least Significant Byte" of count.
5. Numbers below diagrams are count values.
The lower number is the least significant byte.
The upper number is the most significant byte. Since the counter is programmed to Read/Write LSB only, the most significant byte cannot be read.
N stands for an undefined count.
Vertical lines show transitions between count values.

Figure 15. Mode 0

MODE 1: HARDWARE RETRIGGERABLE ONE-SHOT

OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero. OUT will then go high and remain high until the CLK pulse after the next trigger.

After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration. The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected unless the Counter is retriggered. In that case, the Counter is loaded with the new count and the one-shot pulse continues until the new count expires.

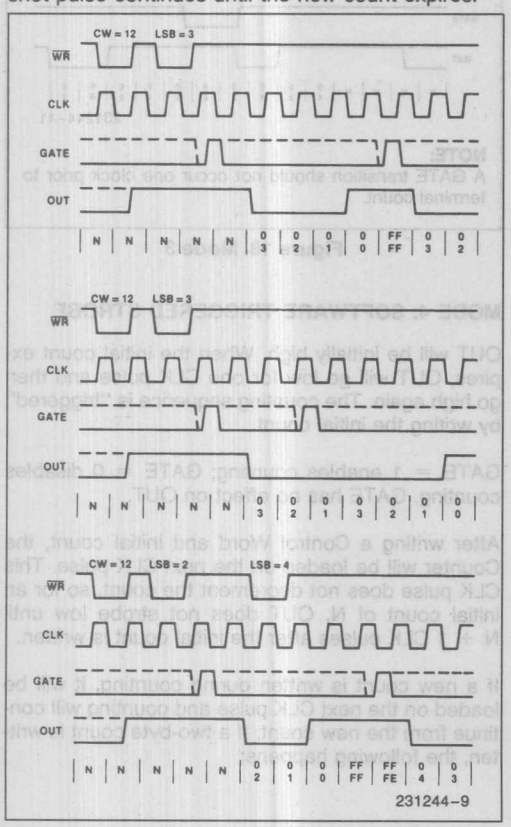


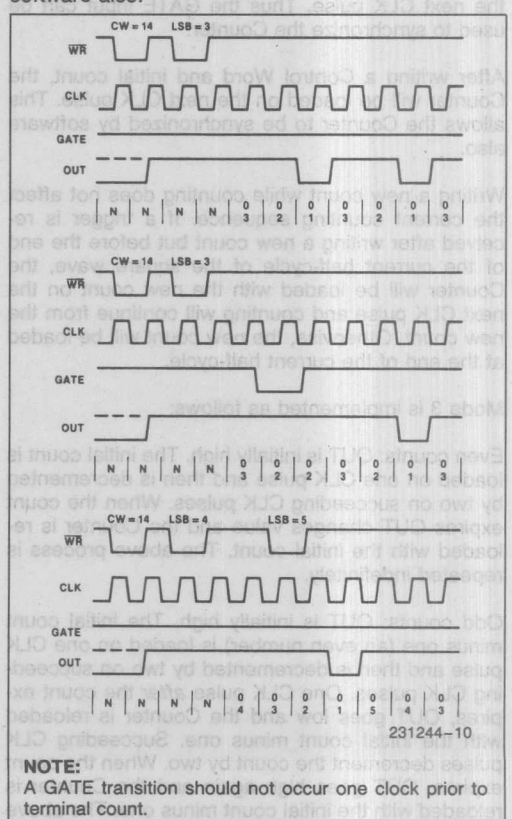
Figure 16. Mode 1

MODE 2: RATE GENERATOR

This Mode functions like a divide-by-N counter. It is typicially used to generate a Real Time Clock interrupt. OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated. Mode 2 is periodic; the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately. A trigger reloads the Counter with the initial count on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written. This allows the Counter to be synchronized by software also.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 17. Mode 2

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.

MODE 3: SQUARE WAVE MODE

Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required. A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

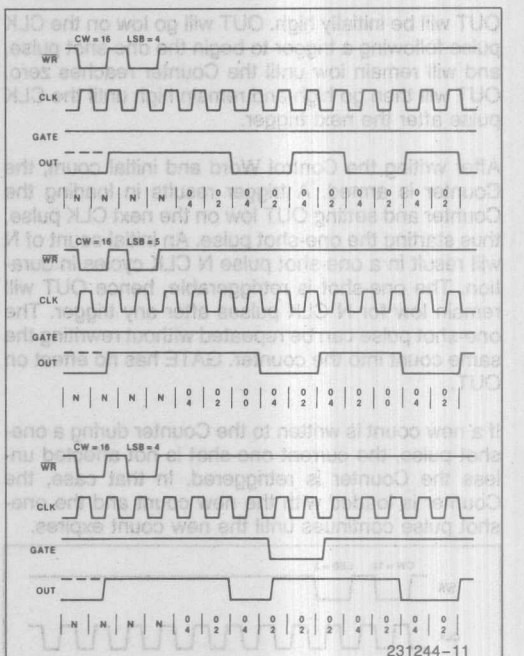
After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

Mode 3 is implemented as follows:

Even counts: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.

Odd counts: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. One CLK pulse *after* the count expires, OUT goes low and the Counter is reloaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT will be high for (N + 1)/2 counts and low for (N − 1)/2 counts.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 18. Mode 3

MODE 4: SOFTWARE TRIGGERED STROBE

OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is "triggered" by writing the initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be "retriggered" by software. OUT strobes low N+1 CLK pulses after the new count of N is written.

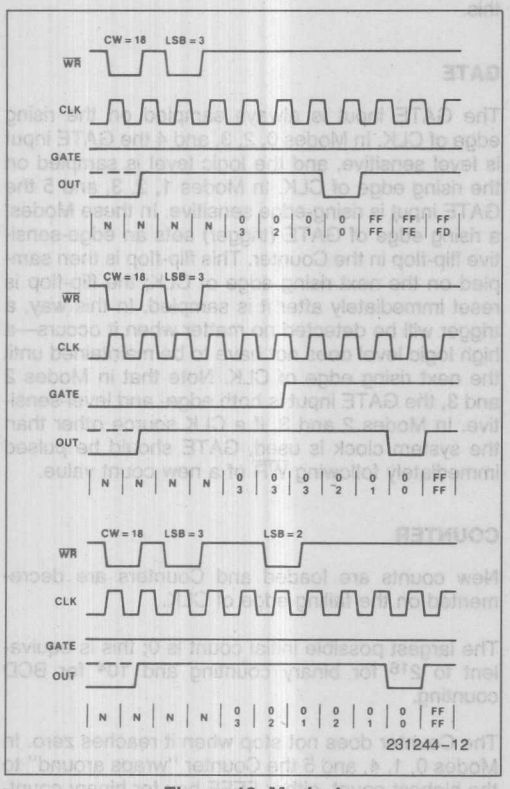


Figure 19. Mode 4

MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N+1 CLK pulses after a trigger.

A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N + 1 CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

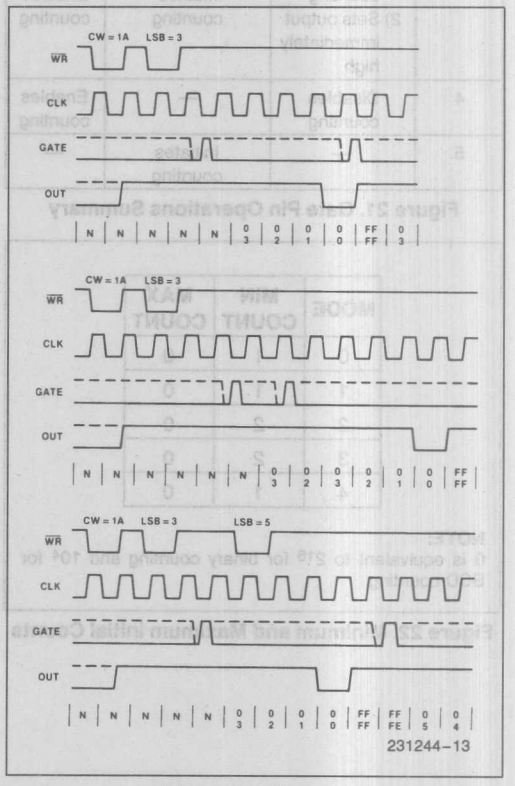


Figure 20. Mode 5

Signal Status Modes	Low Or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets output after next clock	—
2	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

Figure 21. Gate Pin Operations Summary

MODE	MIN COUNT	MAX COUNT
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0

NOTE:
0 is equivalent to 2^{16} for binary counting and 10^4 for BCD counting

Figure 22. Minimum and Maximum initial Counts

Operation Common to All Modes

Programming

When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.

GATE

The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive. In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs—a high logic level does not have to be maintained until the next rising edge of CLK. Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

COUNTER

New counts are loaded and Counters are decremented on the falling edge of CLK.

The largest possible initial count is 0; this is equivalent to 2^{16} for binary counting and 10^4 for BCD counting.

The Counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5 the Counter "wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias.....0°C to 70°C
Storage Temperature .....−65° to +150°C
Supply Voltage .....−0.5 to +8.0V
Operating Voltage .....+4V to +7V
Voltage on any Input.....GND −2V to +6.5V
Voltage on any Output ..GND−0.5V to V_{CC} + 0.5V
Power Dissipation .....1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(T_A=0°C to 70°C, V_{CC}=5V± 10%, GND=0V) (T_A = −40°C to +85°C for Extended Temperature)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC} − 0.4		V V	I_{OH} = −2.5 mA I_{OH} = −100 μA
I_{IL}	Input Load Current		±2.0	μA	V_{IN}=V_{CC} to 0V
I_{OFL}	Output Float Leakage Current		±10	μA	V_{OUT}=V_{CC} to 0.0V
I_{CC}	V_{CC} Supply Current		20	mA	Clk Freq= 8MHz 82C54 10MHz 82C54-2
I_{CCSB}	V_{CC} Supply Current-Standby		10	μA	CLK Freq = DC $\overline{CS}$ = HIGH. All Inputs/Data Bus HIGH All Outputs Floating
I_{CCSB1}	V_{CC} Supply Current-Standby		150	μA	CLK Freq = DC $\overline{CS}$ = HIGH. All Other Inputs, I/O Pins = Low, Outputs Open
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND(5)
C_{OUT}	Output Capacitance		20	pF	

A.C. CHARACTERISTICS

(T_A = 0°C to 70°C, V_{CC} = 5V ±10%, GND =0V) (T_A = −40°C to +85°C for Extended Temperature)

BUS PARAMETERS (Note 1)

READ CYCLE

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{AR}	Address Stable Before $\overline{RD}$ ↓	45		30		ns
t_{SR}	$\overline{CS}$ Stable Before $\overline{RD}$ ↓	0		0		ns
t_{RA}	Address Hold Time After $\overline{RD}$ ↑	0		0		ns
t_{RR}	$\overline{RD}$ Pulse Width	150		95		ns
t_{RD}	Data Delay from $\overline{RD}$ ↓		120		85	ns
t_{AD}	Data Delay from Address		220		185	ns
t_{DF}	$\overline{RD}$ ↑ to Data Floating	5	90	5	65	ns
t_{RV}	Command Recovery Time	200		165		ns

NOTE:
1. AC timings measured at V_{OH} = 2.0V, V_{OL} = 0.8V.

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{AW}	Address Stable Before $\overline{WR}\downarrow$	0		0		ns
t_{SW}	$\overline{CS}$ Stable Before $\overline{WR}\downarrow$	0		0		ns
t_{WA}	Address Hold Time After $\overline{WR}\uparrow$	0		0		ns
t_{WW}	$\overline{WR}$ Pulse Width	150		95		ns
t_{DW}	Data Setup Time Before $\overline{WR}\uparrow$	120		95		ns
t_{WD}	Data Hold Time After $\overline{WR}\uparrow$	0		0		ns
t_{RV}	Command Recovery Time	200		165		ns

CLOCK AND GATE

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{CLK}	Clock Period	125	DC	100	DC	ns
t_{PWH}	High Pulse Width	60(3)		30(3)		ns
t_{PWL}	Low Pulse Width	60(3)		50(3)		ns
T_R	Clock Rise Time		25		25	ns
t_F	Clock Fall Time		25		25	ns
t_{GW}	Gate Width High	50		50		ns
t_{GL}	Gate Width Low	50		50		ns
t_{GS}	Gate Setup Time to CLK $\uparrow$	50		40		ns
t_{GH}	Gate Hold Time After CLK $\uparrow$	50(2)		50(2)		ns
T_{OD}	Output Delay from CLK $\downarrow$		150		100	ns
t_{ODG}	Output Delay from Gate $\downarrow$		120		100	ns
t_{WC}	CLK Delay for Loading(4)	0	55	0	55	ns
t_{WG}	Gate Delay for Sampling(4)	−5	50	−5	40	ns
t_{WO}	OUT Delay from Mode Write		260		240	ns
t_{CL}	CLK Set Up for Count Latch	−40	45	−40	40	ns

NOTES:

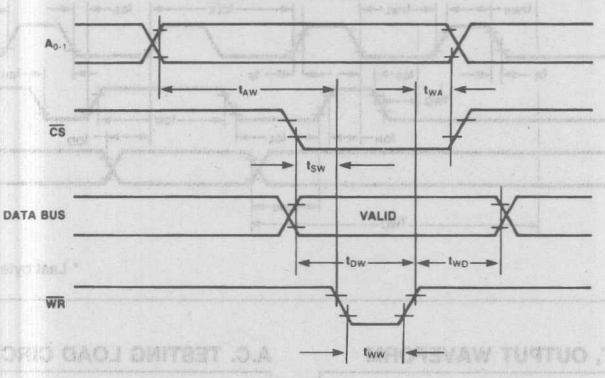
2. In Modes 1 and 5 triggers are sampled on each rising clock edge. A second trigger within 120 ns (70 ns for the 82C54-2) of the rising clock edge may not be detected.
3. Low-going glitches that violate t_{PWH}, t_{PWL} may cause errors requiring counter reprogramming.
4. Except for Extended Temp., See Extended Temp. A.C. Characteristics below.
5. Sampled not 100% tested. $T_A = 25°C$.
6. If CLK present at T_{WC} min then Count equals N+2 CLK pulses, T_{WC} max equals Count N+1 CLK pulse. T_{WC} min to T_{WC} max, count will be either N+1 or N+2 CLK pulses.
7. In Modes 1 and 5, if GATE is present when writing a new Count value, at T_{WG} min Counter will not be triggered, at T_{WG} max Counter will be triggered.
8. If CLK present when writing a Counter Latch or ReadBack Command, at T_{CL} min CLK will be reflected in count value latched, at T_{CL} max CLK will not be reflected in the count value latched. Writing a Counter Latch or ReadBack Command between T_{CL} min and T_{WL} max will result in a latched count vallue which is ± one least significant bit.

EXTENDED TEMPERATURE ($T_A = -40°C$ to $+85°C$ for Extended Temperature)

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{WC}	CLK Delay for Loading	−25	25	−25	25	ns
t_{WG}	Gate Delay for Sampling	−25	25	−25	25	ns

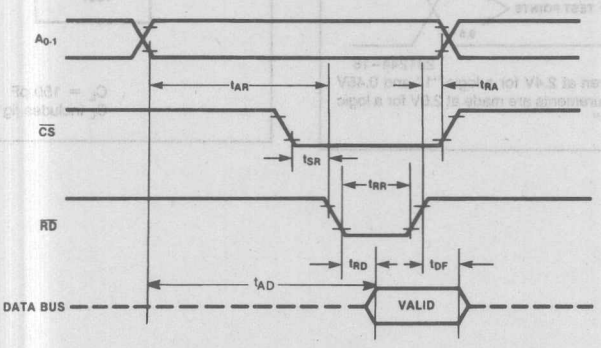
WAVEFORMS

WRITE



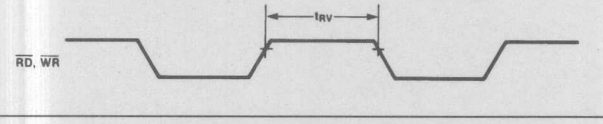
231244-14

READ



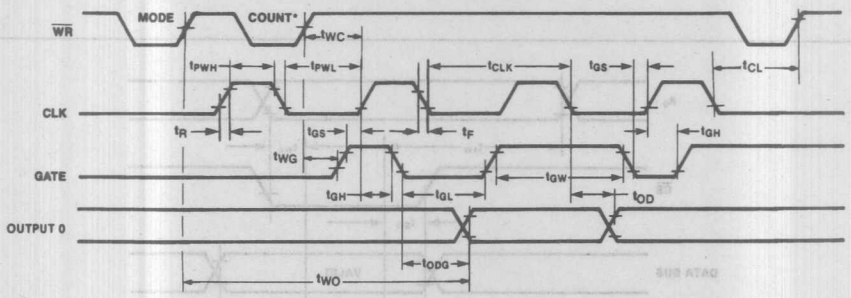
231244-15

RECOVERY



231244-16

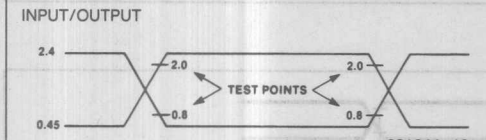
CLOCK AND GATE



231244-17

* Last byte of count being written

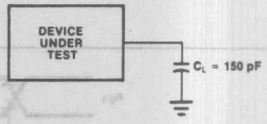
A.C. TESTING INPUT, OUTPUT WAVEFORM



231244-18

A.C. Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

A.C. TESTING LOAD CIRCUIT



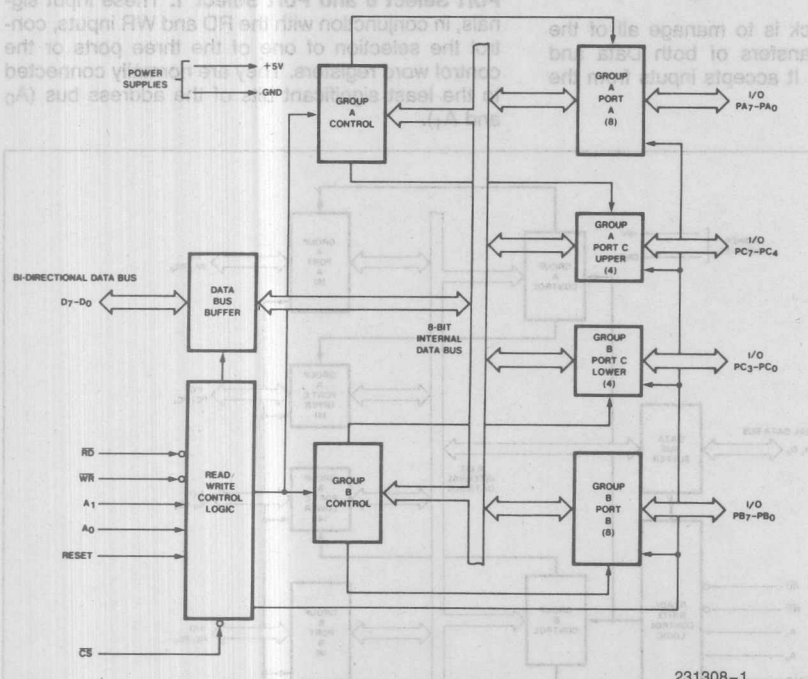
231244-19

C_L = 150 pF
C_L includes jig capacitance

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

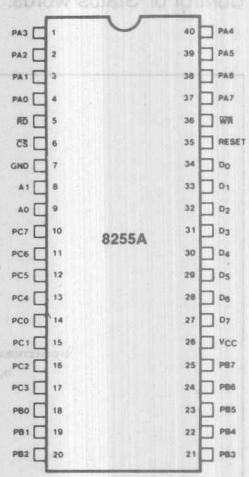
- **MCS-85™ Compatible 8255A-5**
- **24 Programmable I/O Pins**
- **Completely TTL Compatible**
- **Fully Compatible with Intel Microprocessor Families**
- **Improved Timing Characteristics**
- **Direct Bit Set/Reset Capability Easing Control Application Interface**
- **Reduces System Package Count**
- **Improved DC Driving Capability**
- **Available in EXPRESS**
  - **—Standard Temperature Range**
  - **—Extended Temperature Range**
- **40 Pin DIP Package or 44 Lead PLCC**
  (See Intel Packaging: Order Number: 231369)

The Intel 8255A is a general purpose programmable I/O device designed for use with Intel microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.



231308-1

Figure 1. 8255A Block Diagram



231308-2

Figure 2. Pin Configuration

September 1987
Order Number: 231308-002

8255A FUNCTIONAL DESCRIPTION

General

The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 8255A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select. A "low" on this input pin enables the communication between the 8255A and the CPU.

($\overline{RD}$)

Read. A "low" on this input pin enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255A.

($\overline{WR}$)

Write. A "low" on this input pin enables the CPU to write data or control words into the 8255A.

(A_0 and A_1)

Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1).

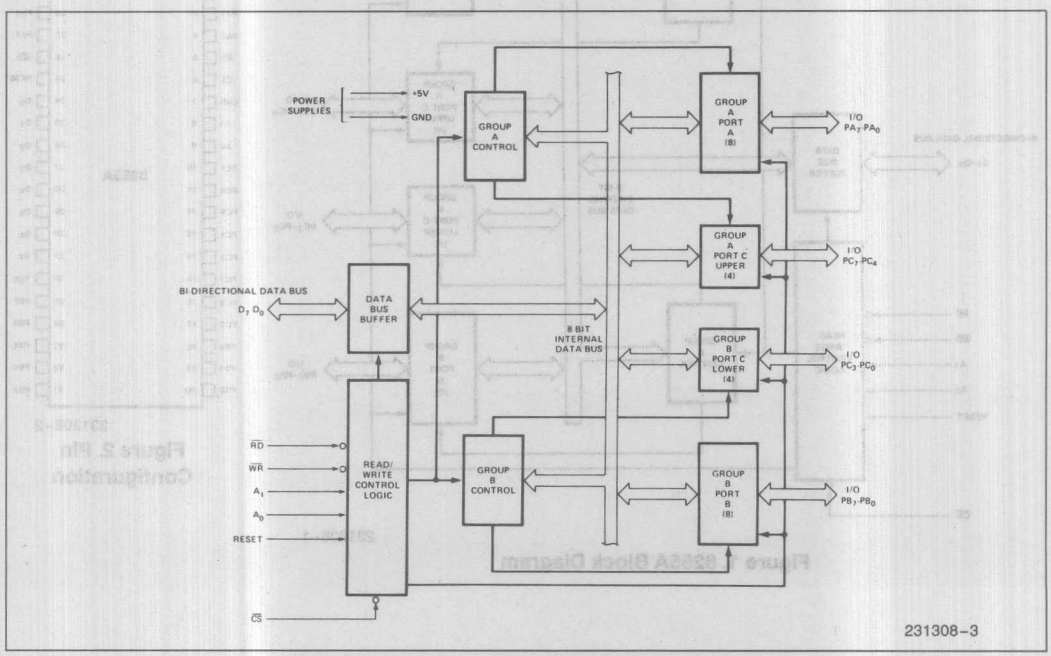


Figure 3. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions

8255A BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input Operation (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
					Output Operation (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
					Disable Function
X	X	X	X	1	Data Bus → 3-State
1	1	0	1	0	Illegal Condition
X	X	1	1	0	Data Bus → 3-State

(RESET)

Reset. A "high" on this input clears the control register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A—Port A and Port C upper (C7–C4)

Control Group B—Port B and Port C lower (C3–C0)

The Control Word Register can **Only** be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

The 8255A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255A.

Port A. One 8-bit data output latch/buffer and one 8-bit data input latch.

Port B. One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

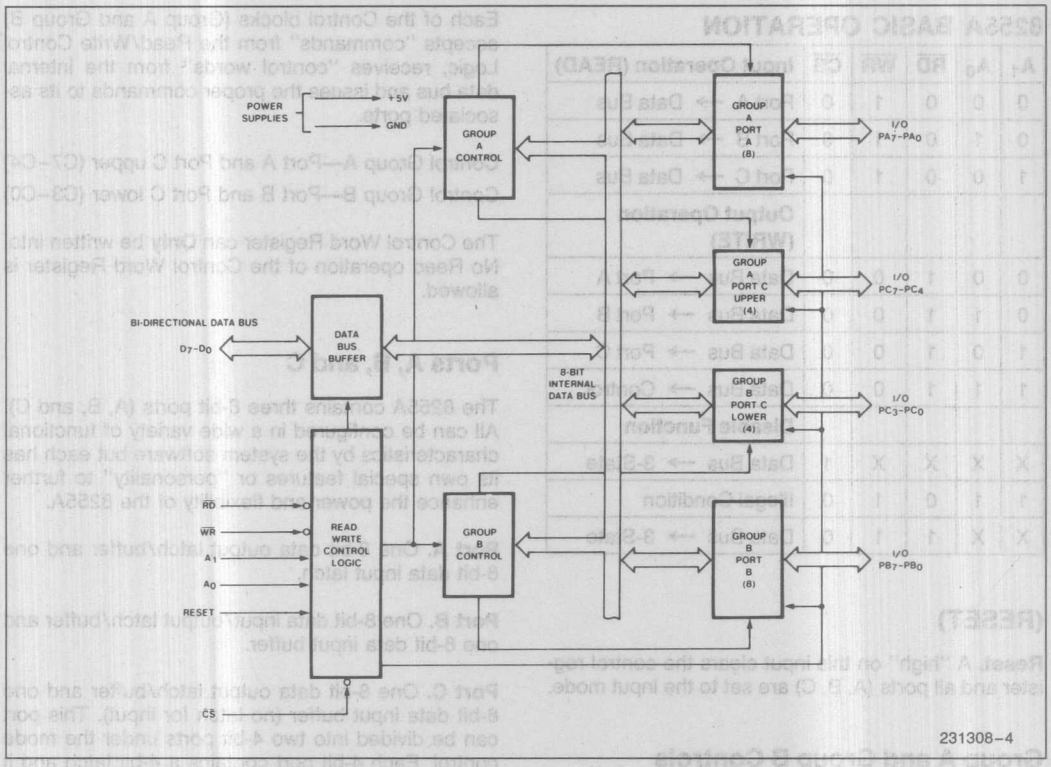
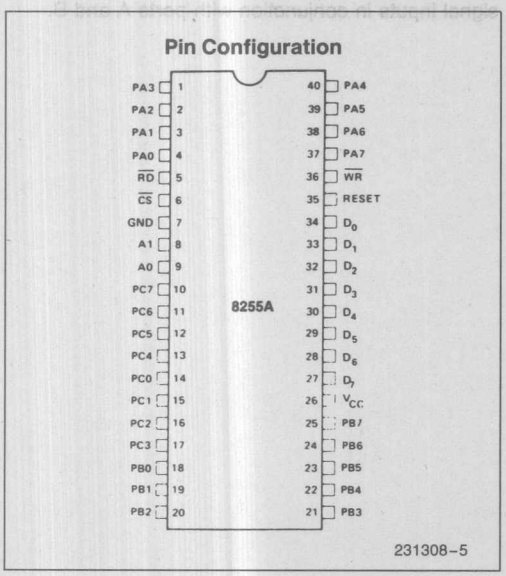


Figure 4. 8225A Block Diagram Showing Group A and Group B Control Functions



Pin Names

D_7–D_0	Data Bus (Bi-Directional)
RESET	Reset Input
$\overline{CS}$	Chip Select
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A0, A1	Port Address
PA7–PA0	Port A (BIT)
PB7–PB0	Port B (BIT)
PC7–PC0	Port C (BIT)
V_{CC}	+5 Volts
GND	0 Volts

8255A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0—Basic Input/Output

Mode 1—Strobed Input/Output

Mode 2—Bi-Directional Bus

When the reset input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the reset is removed the 8255A can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single output instruction. This allows a single 8255A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

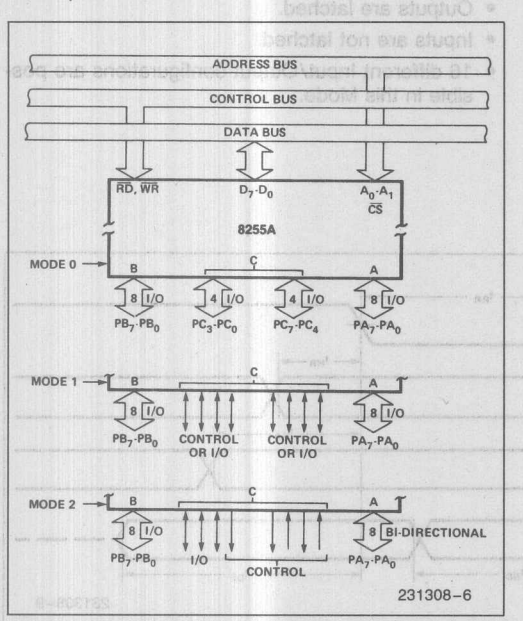


Figure 5. Basic Mode Definitions and Bus Interface

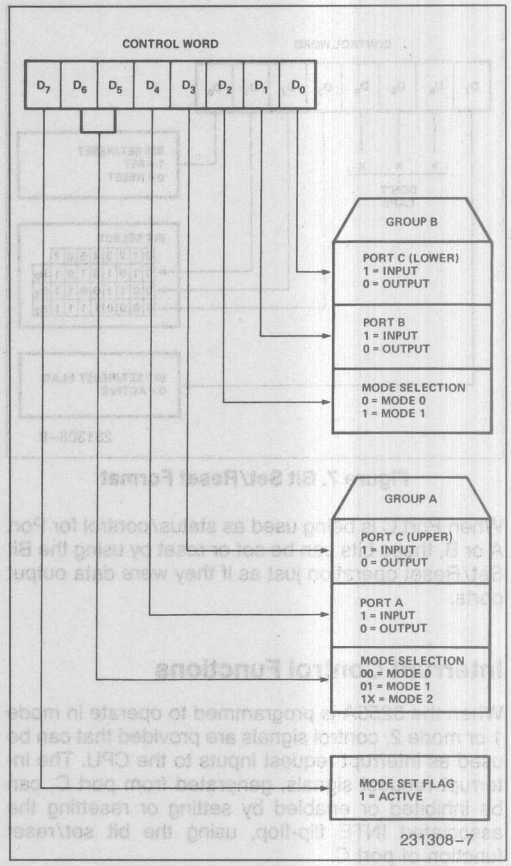


Figure 6. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.

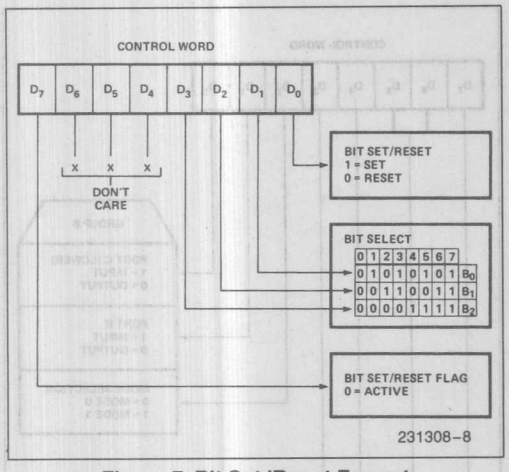


Figure 7. Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET)—INTE is set—Interrupt enable

(BIT-RESET)—INTE is RESET—Interrupt disable

NOTE:
All Mask flip-flops are automatically reset during mode selection and device Reset.

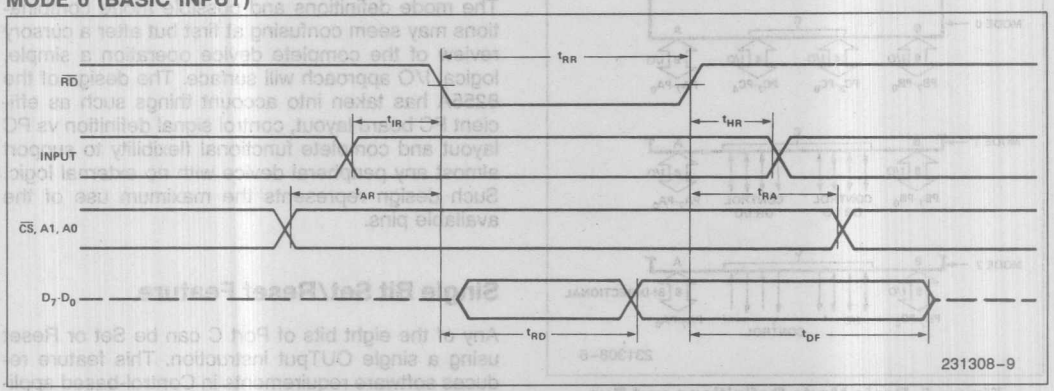
Operating Modes

MODE 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

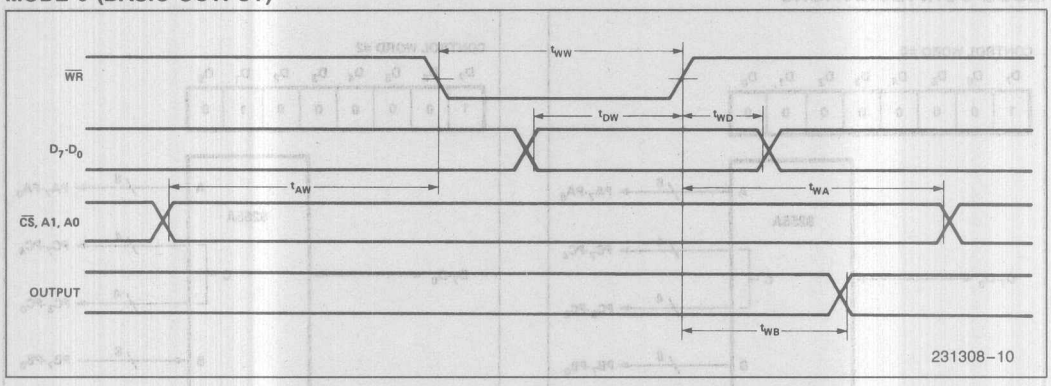
Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

MODE 0 (BASIC INPUT)



MODE 0 (BASIC OUTPUT)



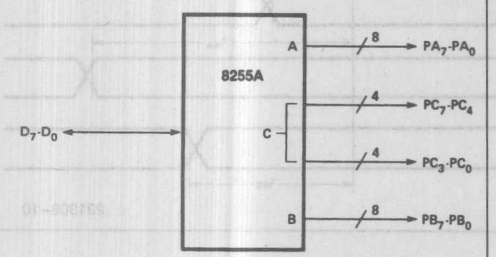
MODE 0 PORT DEFINITION

A		B		Group A			Group B	
D_4	D_3	D_1	D_0	Port A	Port C (Upper)	#	Port B	Port C (Lower)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

MODE CONFIGURATIONS

CONTROL WORD #0

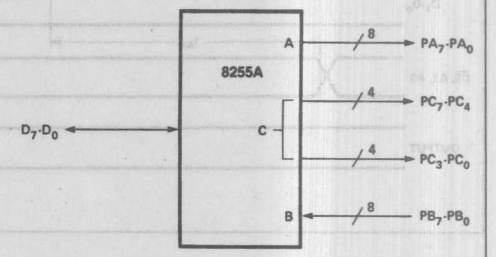
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	0



231308-11

CONTROL WORD #2

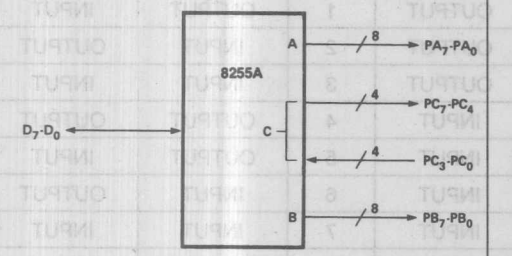
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	0



231308-12

CONTROL WORD #1

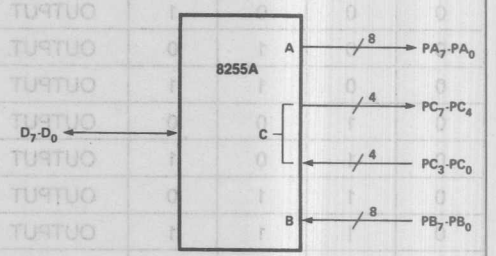
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	1



231308-13

CONTROL WORD #3

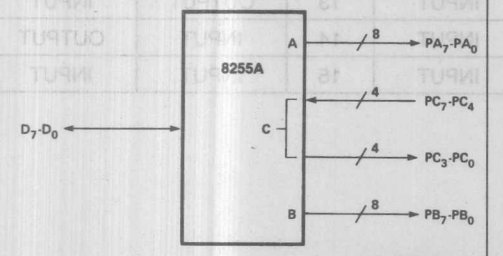
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	1



231308-14

CONTROL WORD #4

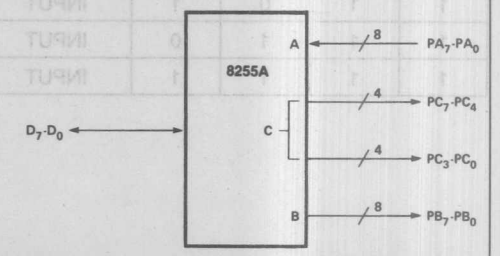
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



231308-15

CONTROL WORD #8

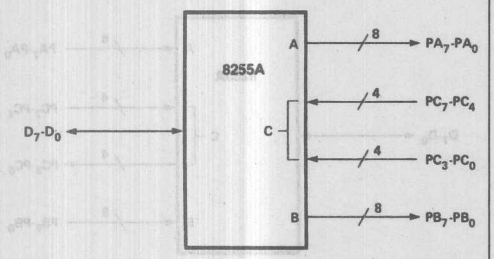
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



231308-16

CONTROL WORD #5

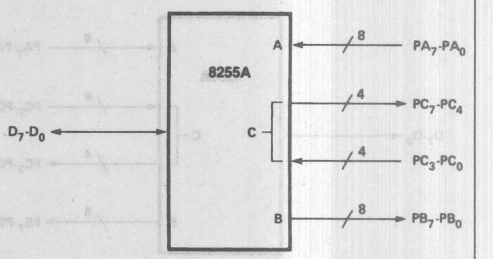
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



231308-17

CONTROL WORD #9

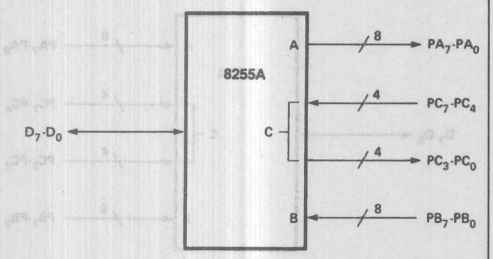
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



231308-18

CONTROL WORD #6

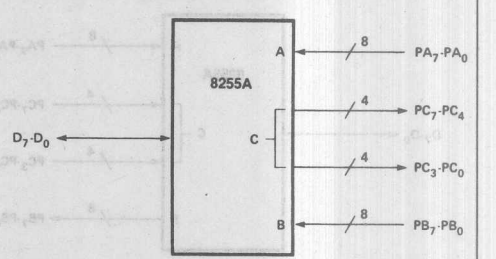
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



231308-19

CONTROL WORD #10

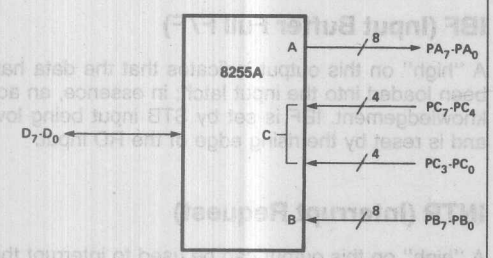
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



231308-20

CONTROL WORD #7

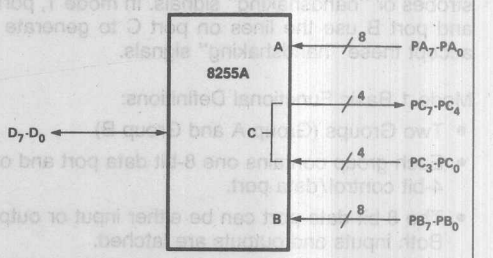
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1



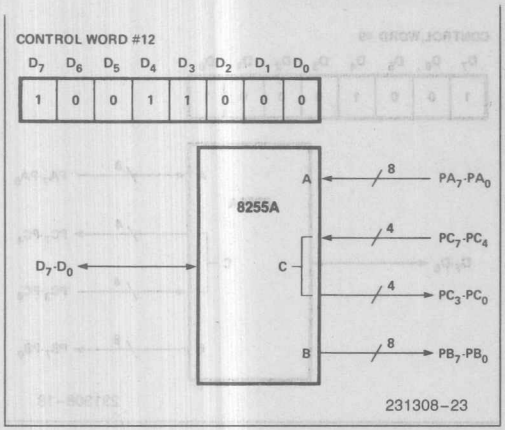
231308-21

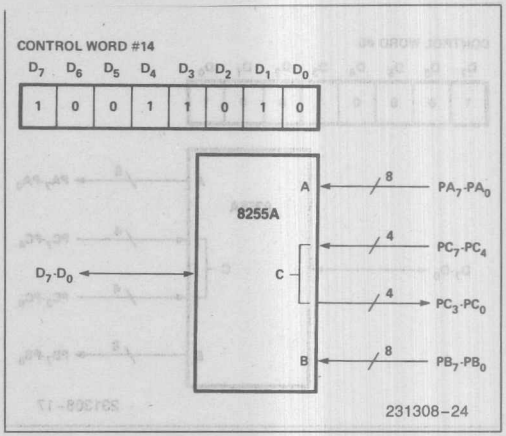
CONTROL WORD #11

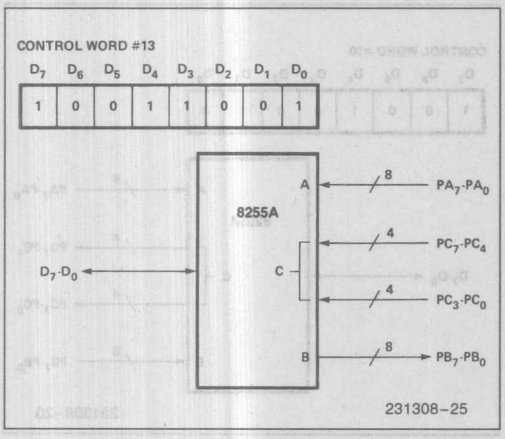
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1

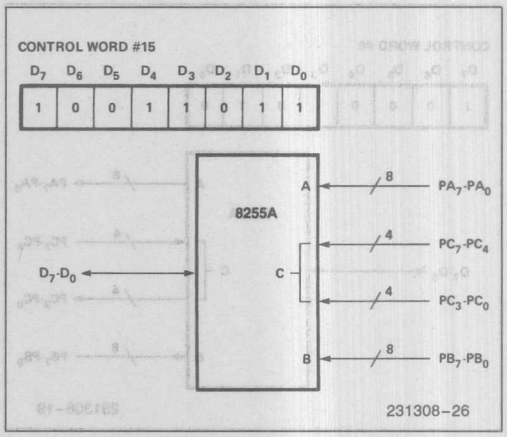


231308-22









Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, port A and port B use the lines on port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

$\overline{\text{STB}}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the $\overline{\text{STB}}$ is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{\text{RD}}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.

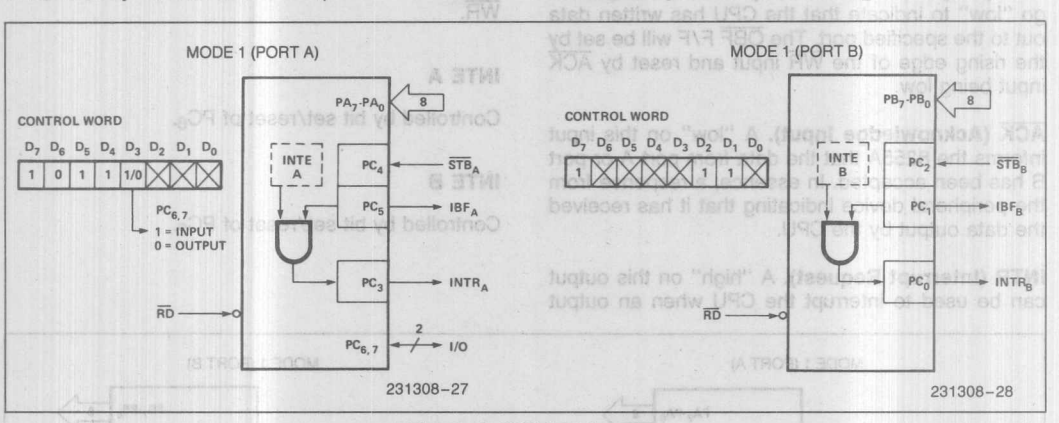


Figure 8. MODE 1 Input

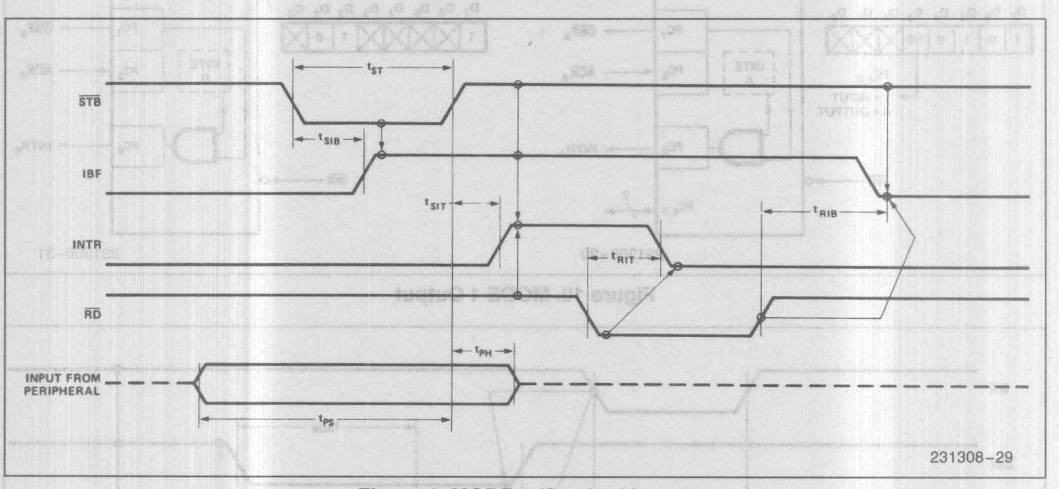


Figure 9. MODE 1 (Strobed Input)

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to the specified port. The $\overline{OBF}$ F/F will be set by the rising edge of the $\overline{WR}$ input and reset by $\overline{ACK}$ input being low.

$\overline{ACK}$ (Acknowledge Input). A "low" on this input informs the 8255A that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a "one", $\overline{OBF}$ is a "one", and INTE is a "one". It is reset by the falling edge of $\overline{WR}$.

INTE A

Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.

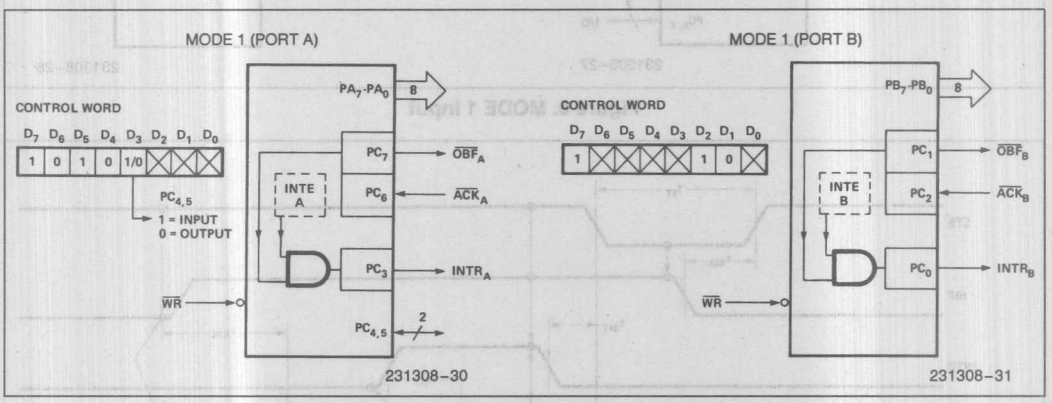


Figure 10. MODE 1 Output

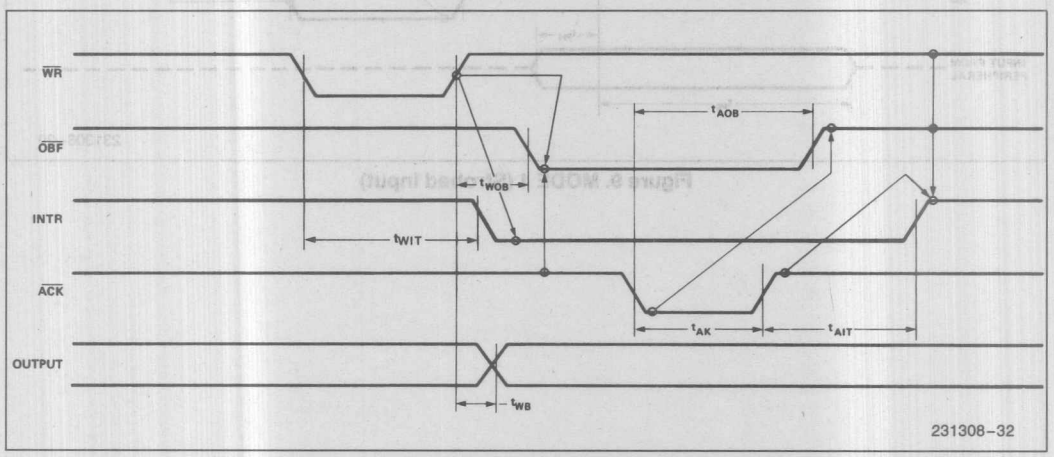


Figure 11. MODE 1 (Strobed Output)

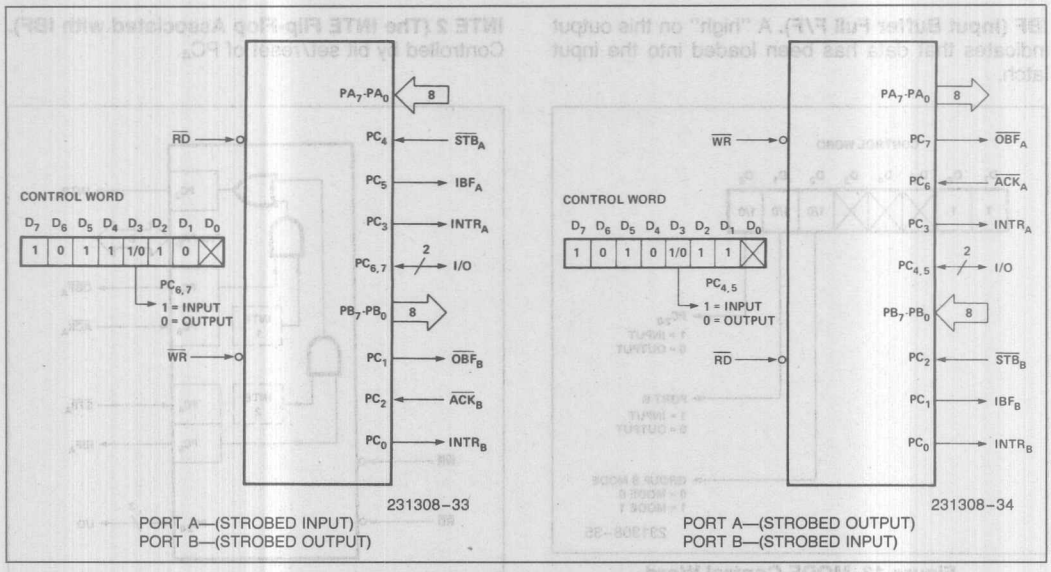


Figure 12. Combinations of MODE 1

Combinations of MODE 1

Port A and Port B can be individually defined as input or output in MODE 1 to support a wide variety of strobed I/O applications.

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A **only**.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{\text{OBF}}$ (Output Buffer Full). The $\overline{\text{OBF}}$ output will go "low" to indicate that the CPU has written data out to port A.

$\overline{\text{ACK}}$ (Acknowledge). A "low" on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (The INTE Flip-Flop Associated with OBF). Controlled by bit set/reset of PC_6.

Input Operations

$\overline{\text{STB}}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

INTE 2 (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC_4.

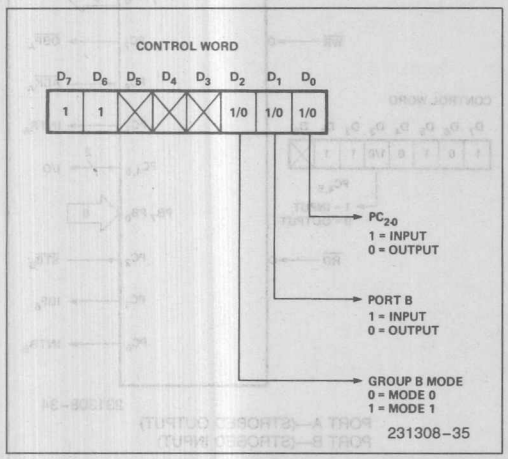


Figure 13. MODE Control Word

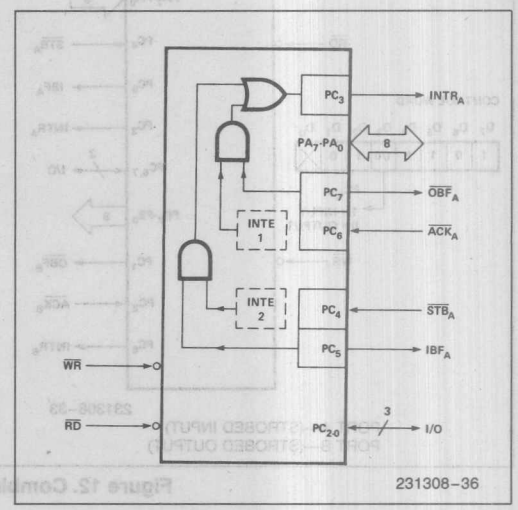
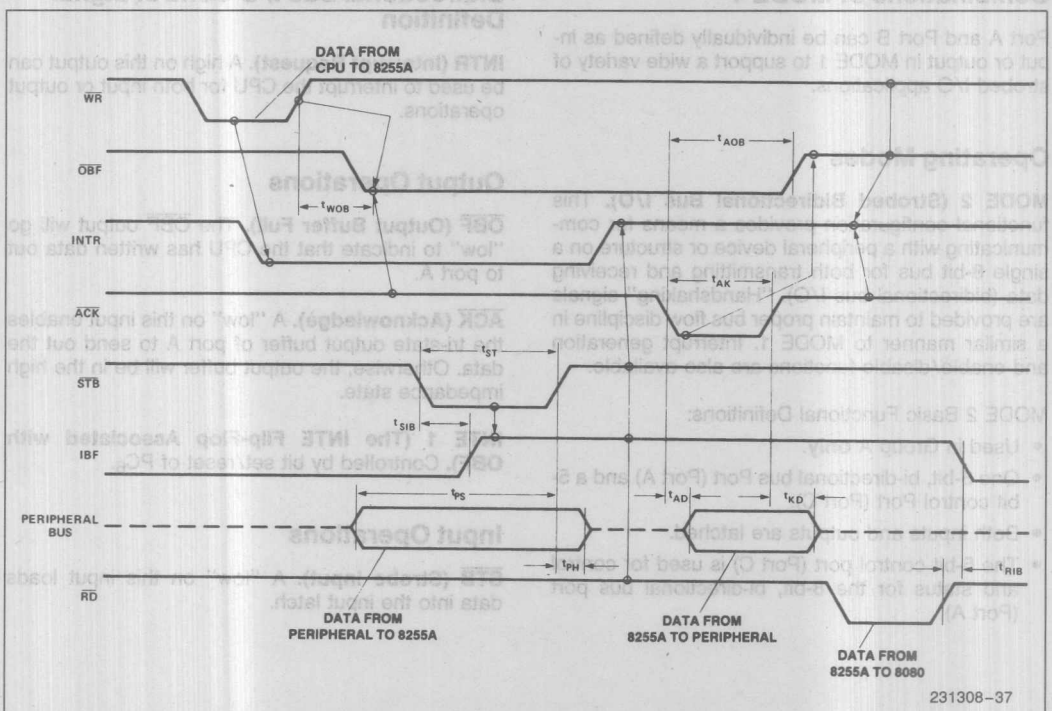


Figure 14. MODE 2



NOTE:
Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
($INTR = IBF \bullet \overline{MASK} \bullet \overline{STB} \bullet \overline{RD} + \overline{OBF} \bullet \overline{MASK} \bullet \overline{ACK} \bullet \overline{WR}$)

Figure 15. MODE 2 (Bidirectional)

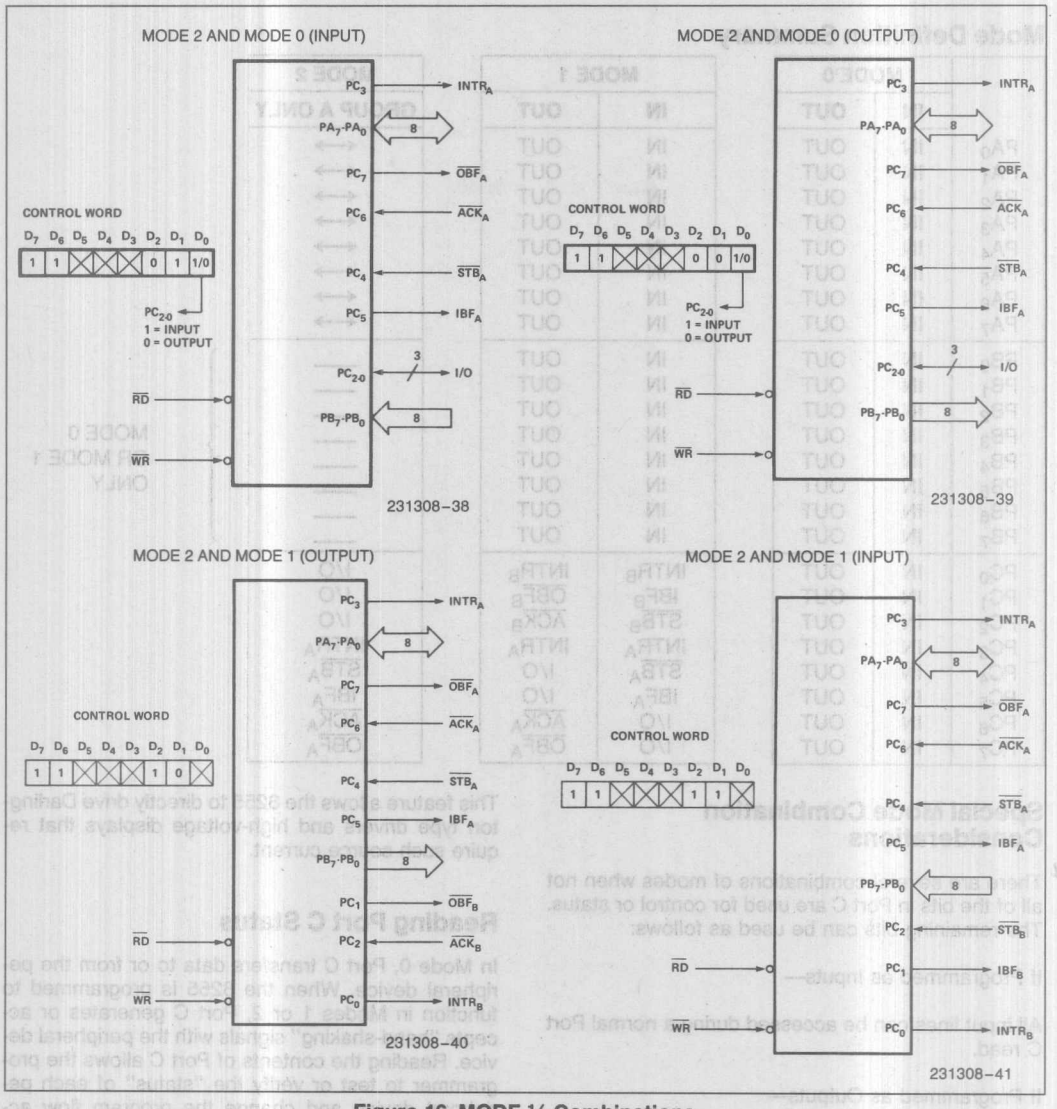


Figure 16. MODE 1/4 Combinations

Mode Definition Summary

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	↔	
PA_1	IN	OUT	IN	OUT	↔	
PA_2	IN	OUT	IN	OUT	↔	
PA_3	IN	OUT	IN	OUT	↔	
PA_4	IN	OUT	IN	OUT	↔	
PA_5	IN	OUT	IN	OUT	↔	
PA_6	IN	OUT	IN	OUT	↔	
PA_7	IN	OUT	IN	OUT	↔	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Special Mode Combination Considerations

There are several combinations of modes when not all of the bits in Port C are used for control or status. The remaining bits can be used as follows:

If Programmed as Inputs—

All input lines can be accessed during a normal Port C read.

If Programmed as Outputs—

Bits in C upper (PC_7–PC_4) must be individually accessed using the bit set/reset function.

Bits in C lower (PC_3–PC_0) can be accessed using the bit set/reset function or accessed as a threesome by writing into Port C.

Source Current Capability on Port B and Port C

Any set of **eight** output buffers, selected randomly from Ports B and C can source 1 mA at 1.5 volts. This feature allows the 8255 to directly drive Darlington type drivers and high-voltage displays that require such source current.

Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 8255 is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

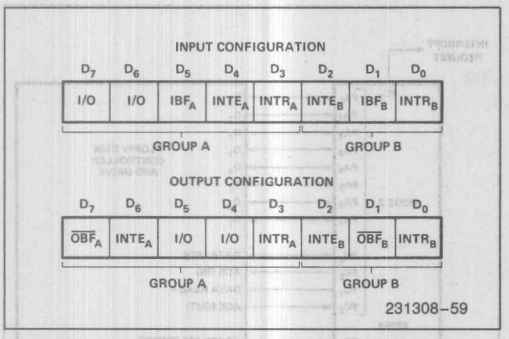


Figure 17. MODE 1 Status Word Format

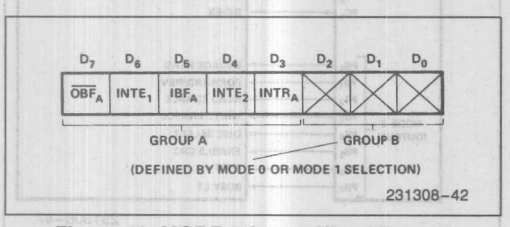


Figure 18. MODE 2 Status Word Format

APPLICATIONS OF THE 8255A

The 8255A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255A is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 8255A to exactly "fit" the application. Figures 19 through 25 represent a few examples of typical applications of the 8255A.

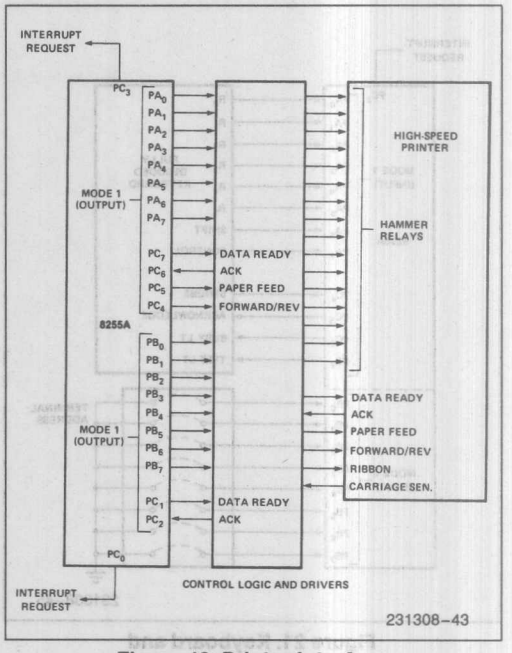


Figure 19. Printer Interface

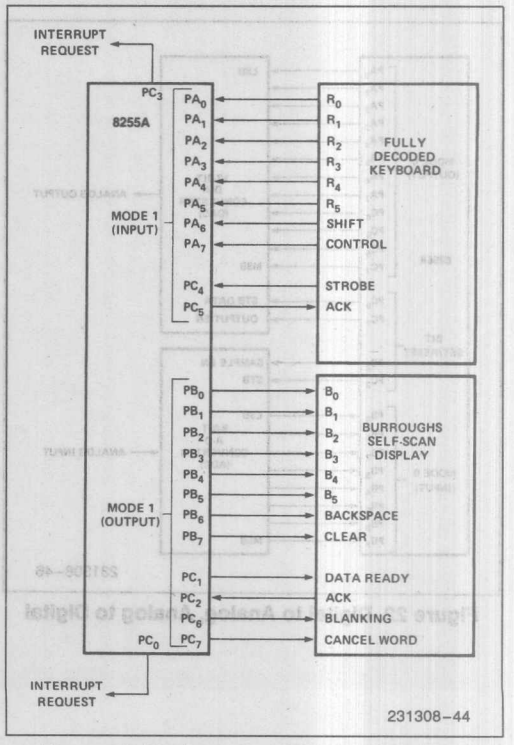


Figure 20. Keyboard and Display Interface

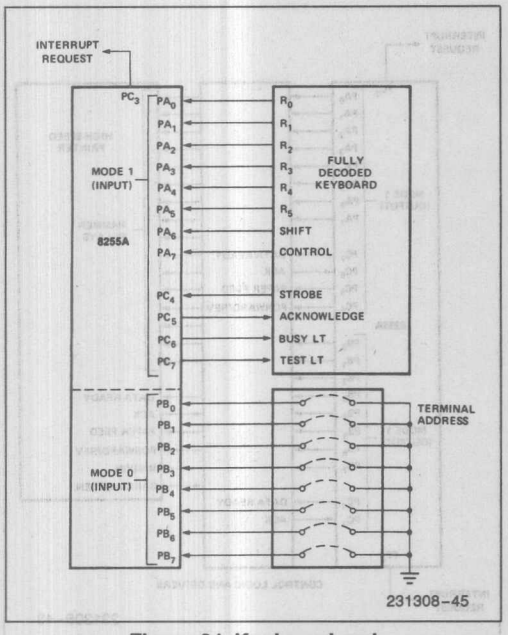


Figure 21. Keyboard and Terminal Address Interface

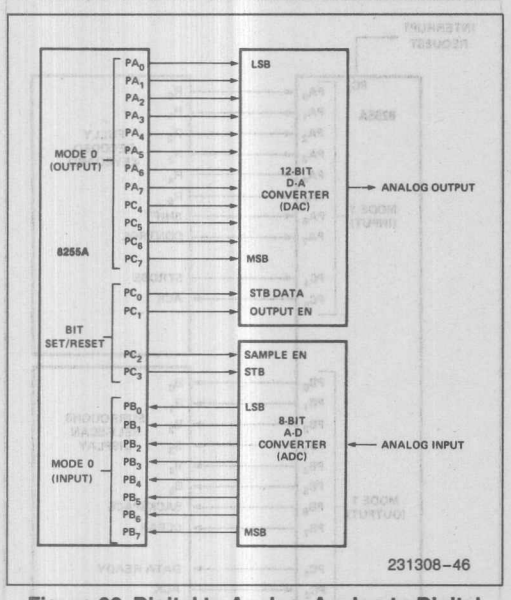


Figure 22. Digital to Analog, Analog to Digital

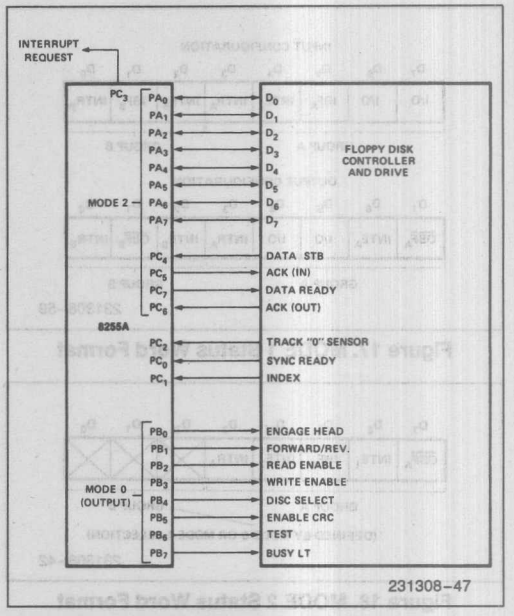


Figure 23. Basic Floppy Disk Interface

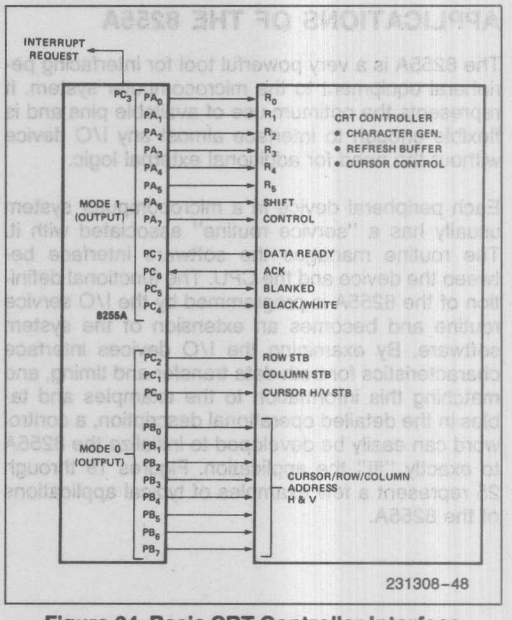


Figure 24. Basic CRT Controller Interface

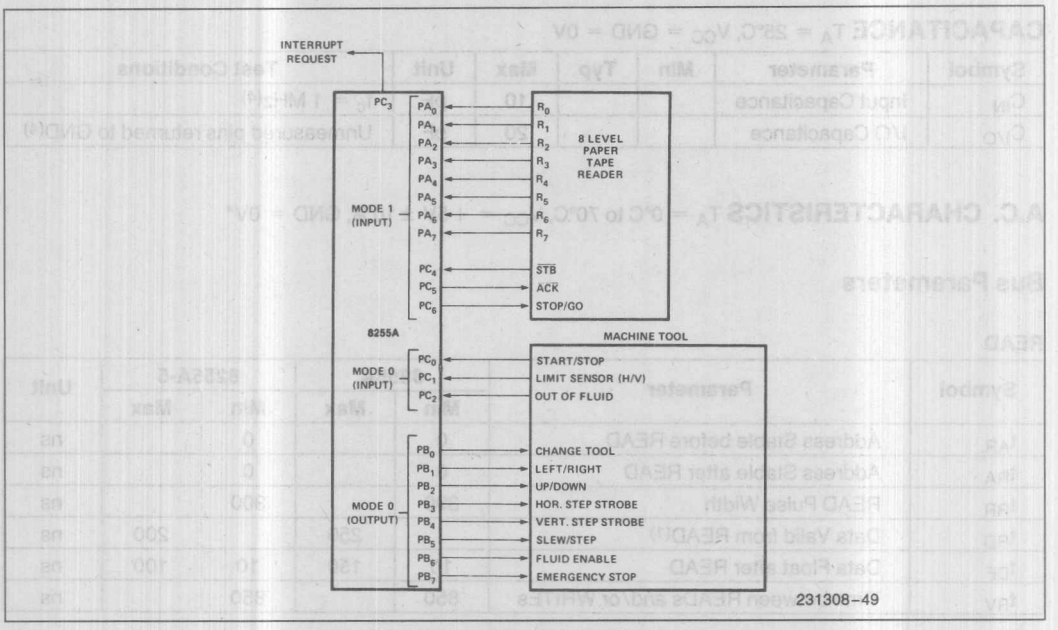


Figure 25. Machine Tool Controller Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with Respect to Ground . . . . . −0.5V to +7V
Power Dissipation . . . . . 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V*

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL} (DB)	Output Low Voltage (Data Bus)		0.45*	V	I_{OL} = 2.5 mA
V_{OL} (PER)	Output Low Voltage (Peripheral Port)		0.45*	V	I_{OL} = 1.7 mA
V_{OH} (DB)	Output High Voltage (Data Bus)	2.4		V	I_{OH} = −400 μA
V_{OH} (PER)	Output High Voltage (Peripheral Port)	2.4		V	I_{OH} = −200 μA
I_{DAR}(1)	Darlington Drive Current	−1.0	−4.0	mA	R_{EXT} = 750Ω; V_{EXT} = 1.5V
I_{CC}	Power Supply Current		120	mA	
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V

NOTE:
1. Available on any 8 pins from Port B and C.

CAPACITANCE $T_A = 25°C$, $V_{CC} = GND = 0V$

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	$f_c = 1$ MHz(4)
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND(4)

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = +5V \pm 10\%$, GND = 0V*

Bus Parameters

READ

Symbol	Parameter	8255A		8255A-5		Unit
		Min	Max	Min	Max	
t_{AR}	Address Stable before READ	0		0		ns
t_{RA}	Address Stable after READ	0		0		ns
t_{RR}	READ Pulse Width	300		300		ns
t_{RD}	Data Valid from READ(1)		250		200	ns
t_{DF}	Data Float after READ	10	150	10	100	ns
t_{RV}	Time between READs and/or WRITEs	850		850		ns

WRITE

Symbol	Parameter	8255A		8255A-5		Unit
		Min	Max	Min	Max	
t_{AW}	Address Stable before WRITE	0		0		ns
t_{WA}	Address Stable after WRITE	20		20		ns
t_{WW}	WRITE Pulse Width	400		300		ns
t_{DW}	Data Valid to WRITE (T.E.)	100		100		ns
t_{WD}	Data Valid after WRITE	30		30		ns

OTHER TIMINGS

Symbol	Parameter	8255A		8255A-5		Unit
		Min	Max	Min	Max	
t_{WB}	WR = 1 to Output(1)		350		350	ns
t_{IR}	Peripheral Data before RD	0		0		ns
t_{HR}	Peripheral Data after RD	0		0		ns
t_{AK}	ACK Pulse Width	300		300		ns
t_{ST}	STB Pulse Width	500		500		ns
t_{PS}	Per. Data before T.E. of STB	0		0		ns
t_{PH}	Per. Data after T.E. of STB	180		180		ns
t_{AD}	ACK = 0 to Output(1)		300		300	ns
t_{KD}	ACK = 1 to Output Float	20	250	20	250	ns

A.C. CHARACTERISTICS (Continued)

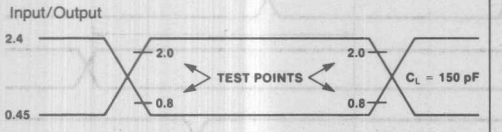
OTHER TIMINGS (Continued)

Symbol	Parameter	8255A		8255A-5		Unit
		Min	Max	Min	Max	
t_{WOB}	WR = 1 to OBF = 0(1)		650		650	ns
t_{AOB}	ACK = 0 to OBF = 1(1)		350		350	ns
t_{SIB}	STB = 0 to IBF = 1(1)		300		300	ns
t_{RIB}	RD = 1 to IBF = 0(1)		300		300	ns
t_{RIT}	RD = 0 to INTR = 0(1)		400		400	ns
t_{SIT}	STB = 1 to INTR = 1(1)		300		300	ns
t_{AIT}	ACK = 1 to INTR = 1(1)		350		350	ns
t_{WIT}	WR = 0 to INTR = 0(1, 3)		850		850	ns

NOTES:
1. Test Conditions: C_L = 150 pF.
2. Period of Reset pulse must be at least 50 μs during or after power on. Subsequent Reset pulse can be 500 ns min.
3. INTR ↑ may occur as early as $\overline{WR}$ ↓.
4. Sampled, not 100% tested.

*For Extended Temperature EXPRESS, use M8255A electrical parameters.

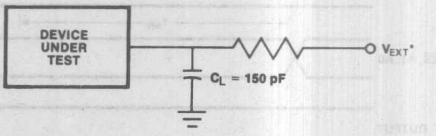
A.C. TESTING INPUT, OUTPUT WAVEFORM



231308-50

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT

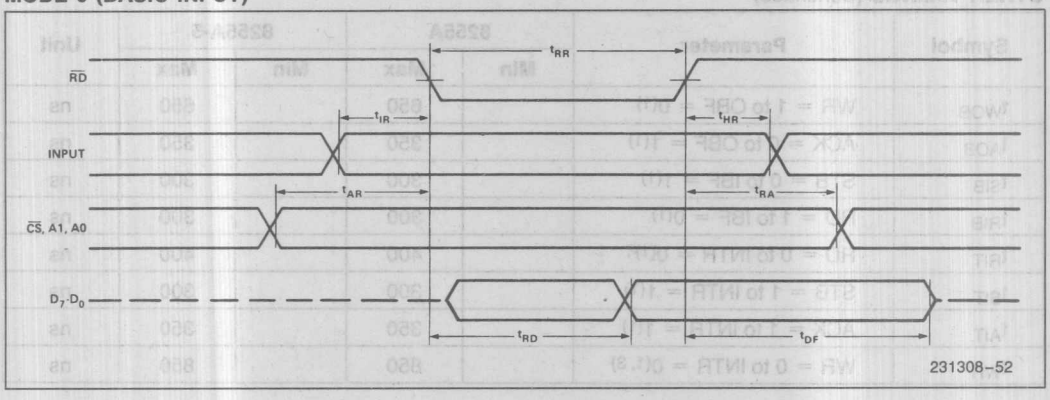


231308-51

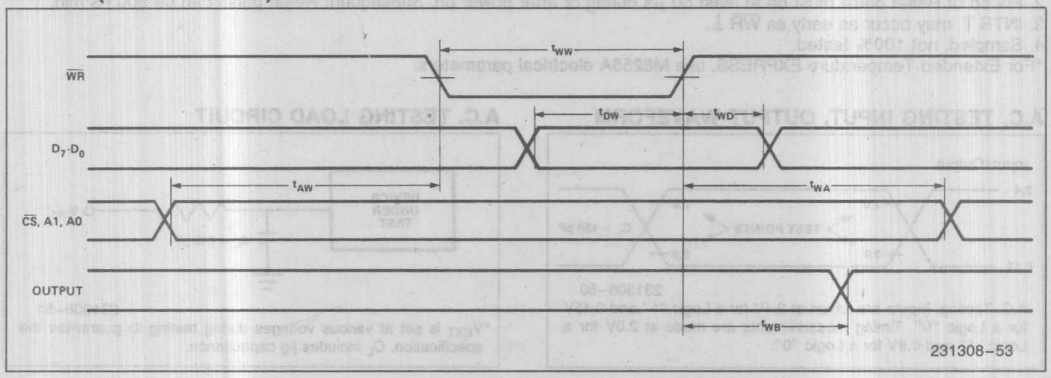
*V_{EXT} is set at various voltages during testing to guarantee the specification. C_L includes jig capacitance.

WAVEFORMS

MODE 0 (BASIC INPUT)

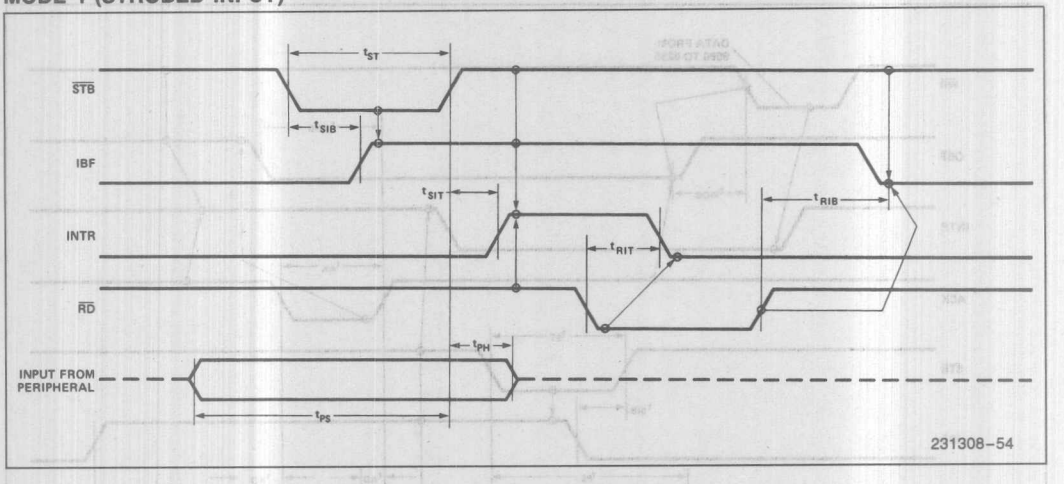


MODE 0 (BASIC OUTPUT)

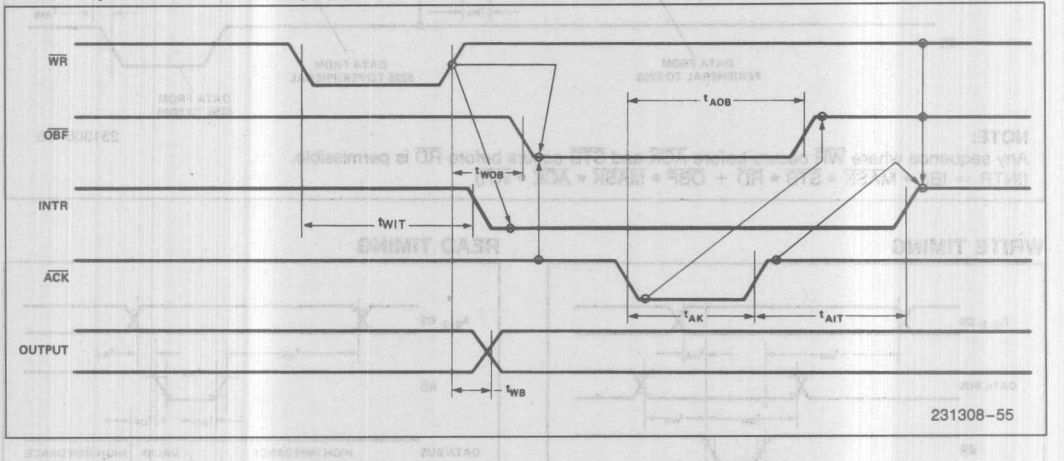


WAVEFORMS (Continued)

MODE 1 (STROBED INPUT)

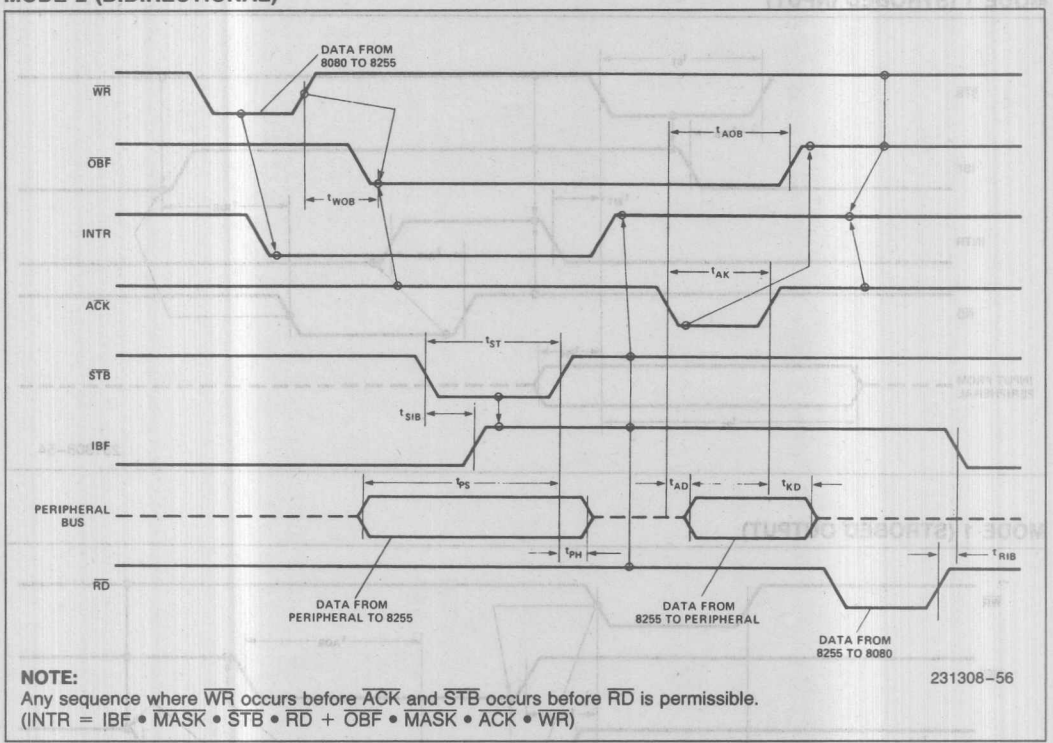


MODE 1 (STROBED OUTPUT)



WAVEFORMS (Continued)

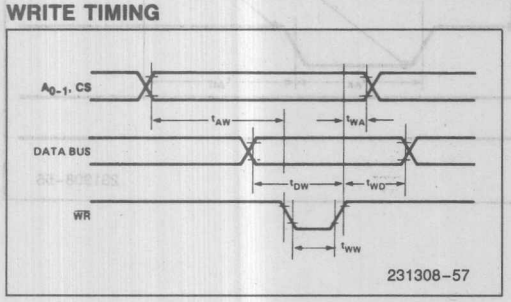
MODE 2 (BIDIRECTIONAL)



231308-56

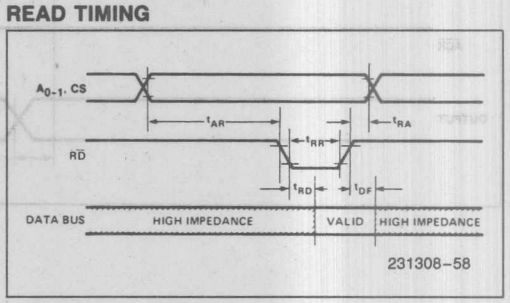
NOTE:
Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
($INTR = IBF \bullet \overline{MASK} \bullet \overline{STB} \bullet \overline{RD} + \overline{OBF} \bullet \overline{MASK} \bullet \overline{ACK} \bullet \overline{WR}$)

WRITE TIMING



231308-57

READ TIMING



231308-58

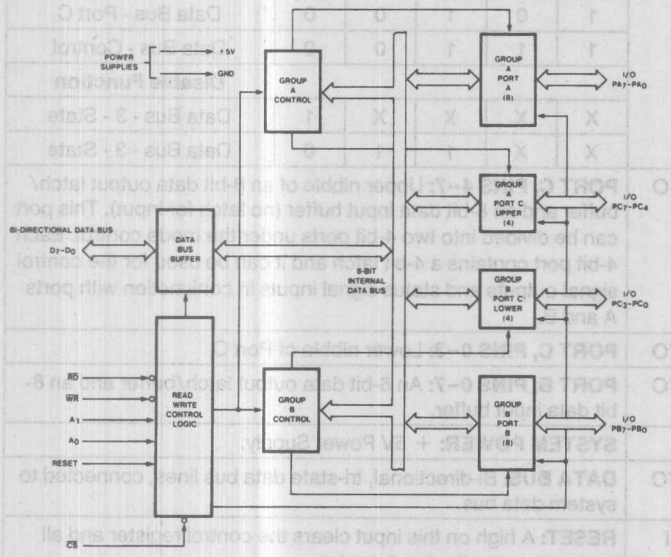
82C55A
CHMOS PROGRAMMABLE PERIPHERAL INTERFACE

- **Compatible with all Intel and Most Other Microprocessors**
- **High Speed, "Zero Wait State" Operation with 8 MHz 8086/88 and 80186/188**
- **24 Programmable I/O Pins**
- **Low Power CHMOS**
- **Completely TTL Compatible**
- **Control Word Read-Back Capability**
- **Direct Bit Set/Reset Capability**
- **2.5 mA DC Drive Capability on all I/O Port Outputs**
- **Available in 40-Pin DIP and 44-Pin PLCC**
- **Available in EXPRESS**
  - **—Standard Temperature Range**
  - **—Extended Temperature Range**

The Intel 82C55A is a high-performance, CHMOS version of the industry standard 8255A general purpose programmable I/O device which is designed for use with all Intel and most other microprocessors. It provides 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. The 82C55A is pin compatible with the NMOS 8255A and 8255A-5.

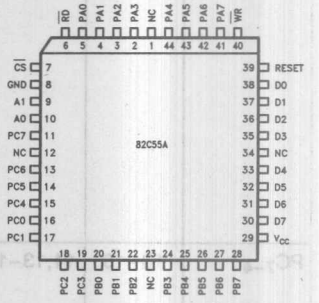
In MODE 0, each group of 12 I/O pins may be programmed in sets of 4 and 8 to be inputs or outputs. In MODE 1, each group may be programmed to have 8 lines of input or output. 3 of the remaining 4 pins are used for handshaking and interrupt control signals. MODE 2 is a strobed bi-directional bus configuration.

The 82C55A is fabricated on Intel's advanced CHMOS III technology which provides low power consumption with performance equal to or greater than the equivalent NMOS product. The 82C55A is available in 40-pin DIP and 44-pin plastic leaded chip carrier (PLCC) packages.

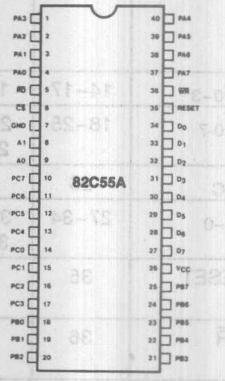


231256-1

Figure 1. 82C55A Block Diagram



231256-31



231256-2

Figure 2. 82C55A Pinout

Diagrams are for pin reference only. Package sizes are not to scale.

September 1987
Order Number: 231256-004

Table 1. Pin Description

Symbol	Pin Number Dip	Pin Number PLCC	Type	Name and Function
PA_{3-0}	1–4	2–5	I/O	**PORT A, PINS 0–3:** Lower nibble of an 8-bit data output latch/buffer and an 8-bit data input latch.
$\overline{RD}$	5	6	I	**READ CONTROL:** This input is low during CPU read operations.
$\overline{CS}$	6	7	I	**CHIP SELECT:** A low on this input enables the 82C55A to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and WR are ignored otherwise.
GND	7	8		**System Ground**
A_{1-0}	8–9	9–10	I	**ADDRESS:** These input signals, in conjunction $\overline{RD}$ and $\overline{WR}$, control the selection of one of the three ports or the control word registers.

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input Operation (Read)
0	0	0	1	0	Port A - Data Bus
0	1	0	1	0	Port B - Data Bus
1	0	0	1	0	Port C - Data Bus
1	1	0	1	0	Control Word - Data Bus
					Output Operation (Write)
0	0	1	0	0	Data Bus - Port A
0	1	1	0	0	Data Bus - Port B
1	0	1	0	0	Data Bus - Port C
1	1	1	0	0	Data Bus - Control
					Disable Function
X	X	X	X	1	Data Bus - 3 - State
X	X	1	1	0	Data Bus - 3 - State

PC_{7-4}	10–13	11,13–15	I/O	**PORT C, PINS 4–7:** Upper nibble of an 8-bit data output latch/buffer and an 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.
PC_{0-3}	14–17	16–19	I/O	**PORT C, PINS 0–3:** Lower nibble of Port C.
PB_{0-7}	18–25	20–22, 24–28	I/O	**PORT B, PINS 0–7:** An 8-bit data output latch/buffer and an 8-bit data input buffer.
V_{CC}	26	29		**SYSTEM POWER:** + 5V Power Supply.
D_{7-0}	27–34	30–33, 35–38	I/O	**DATA BUS:** Bi-directional, tri-state data bus lines, connected to system data bus.
RESET	35	39	I	**RESET:** A high on this input clears the control register and all ports are set to the input mode.
$\overline{WR}$	36	40	I	**WRITE CONTROL:** This input is low during CPU write operations.
PA_{7-4}	37–40	41–44	I/O	**PORT A, PINS 4–7:** Upper nibble of an 8-bit data output latch/buffer and an 8-bit data input latch.
NC		1, 12, 23, 34		No Connect

82C55A FUNCTIONAL DESCRIPTION

General

The 82C55A is a programmable peripheral interface device designed for use in Intel microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 82C55A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 82C55A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7–C4)
Control Group B - Port B and Port C lower (C3–C0)

The control word register can be both written and read as shown in the address decode table in the pin descriptions. Figure 6 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic "1", as this implies control word mode information.

Ports A, B, and C

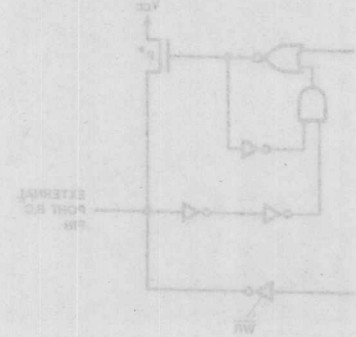
The 82C55A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 82C55A.

Port A. One 8-bit data output latch/buffer and one 8-bit input latch buffer. Both "pull-up" and "pull-down" bus hold devices are present on Port A.

Port B. One 8-bit data input/output latch/buffer. Only "pull-up" bus hold devices are present on Port B.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B. Only "pull-up" bus hold devices are present on Port C.

See Figure 4 for the bus-hold circuit configuration for Port A, B, and C.



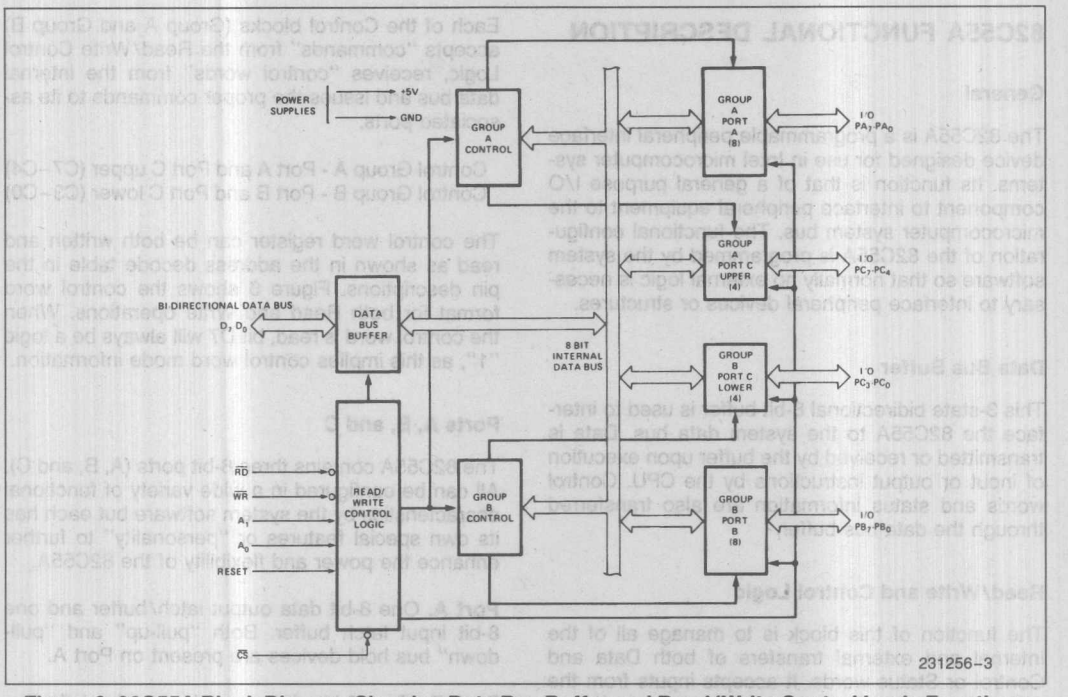
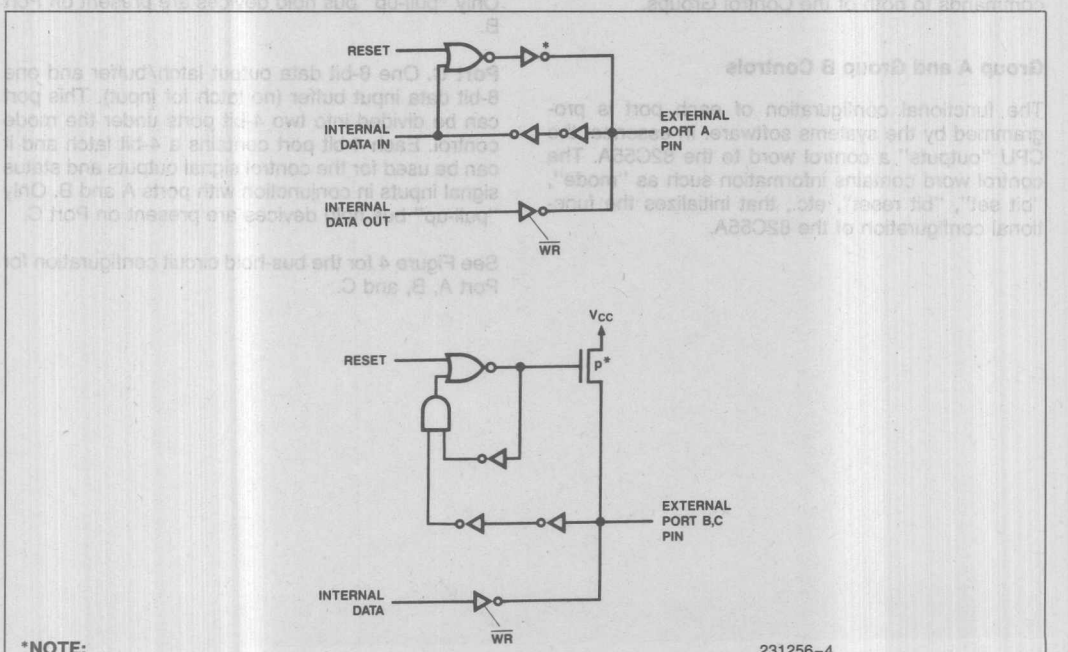


Figure 3. 82C55A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions



***NOTE:**
Port pins loaded with more than 20 pF capacitance may not have their logic level guaranteed following a hardware reset.

Figure 4. Port A, B, C, Bus-hold Configuration

82C55A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 — Basic input/output
Mode 1 — Strobed Input/output
Mode 2 — Bi-directional Bus

When the reset input goes "high" all ports will be set to the input mode with all 24 port lines held at a logic "one" level by the internal bus hold devices (see Figure 4 Note). After the reset is removed the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need for pullup or pulldown devices in "all CMOS" designs. During the execution of the system program, any of the other modes may be selected by using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

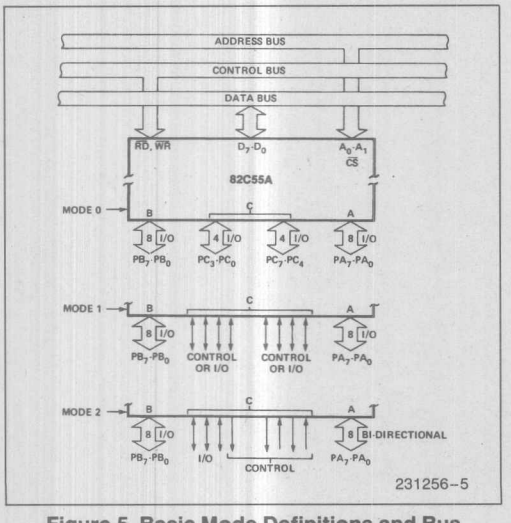


Figure 5. Basic Mode Definitions and Bus Interface

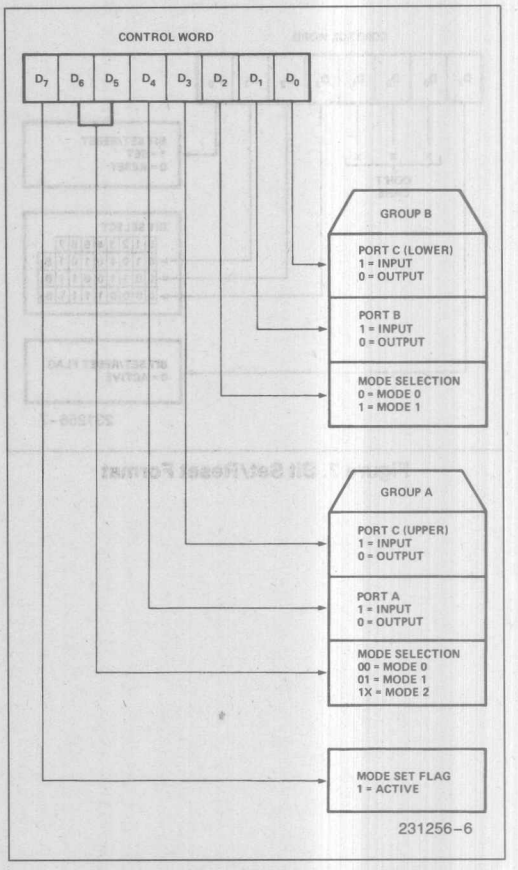


Figure 6. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 82C55A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

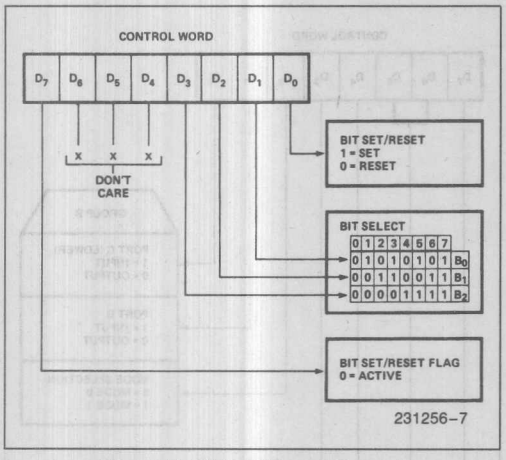


Figure 7. Bit Set/Reset Format

Interrupt Control Functions

When the 82C55A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET)—INTE is SET—Interrupt enable
(BIT-RESET)—INTE is RESET—Interrupt disable

Note:
All Mask flip-flops are automatically reset during mode selection and device Reset.

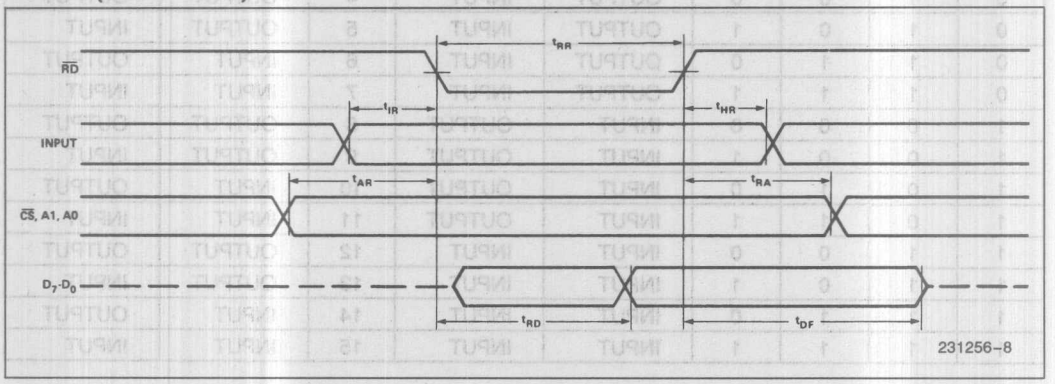
Operating Modes

Mode 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

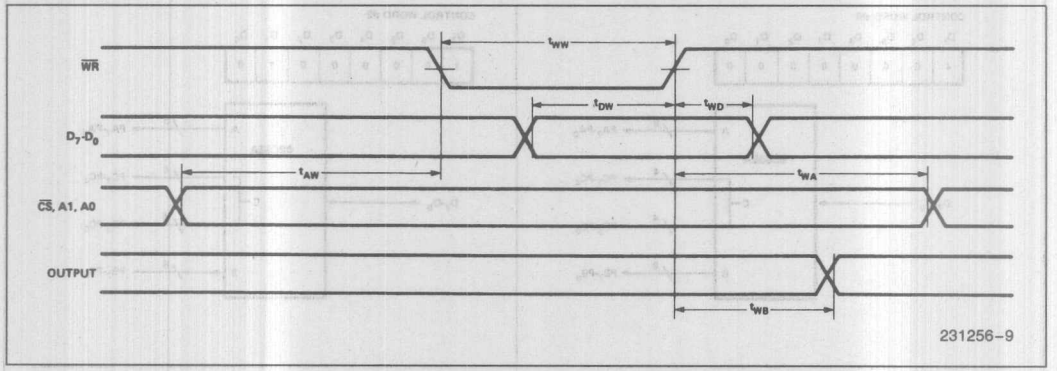
Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

MODE 0 (BASIC INPUT)



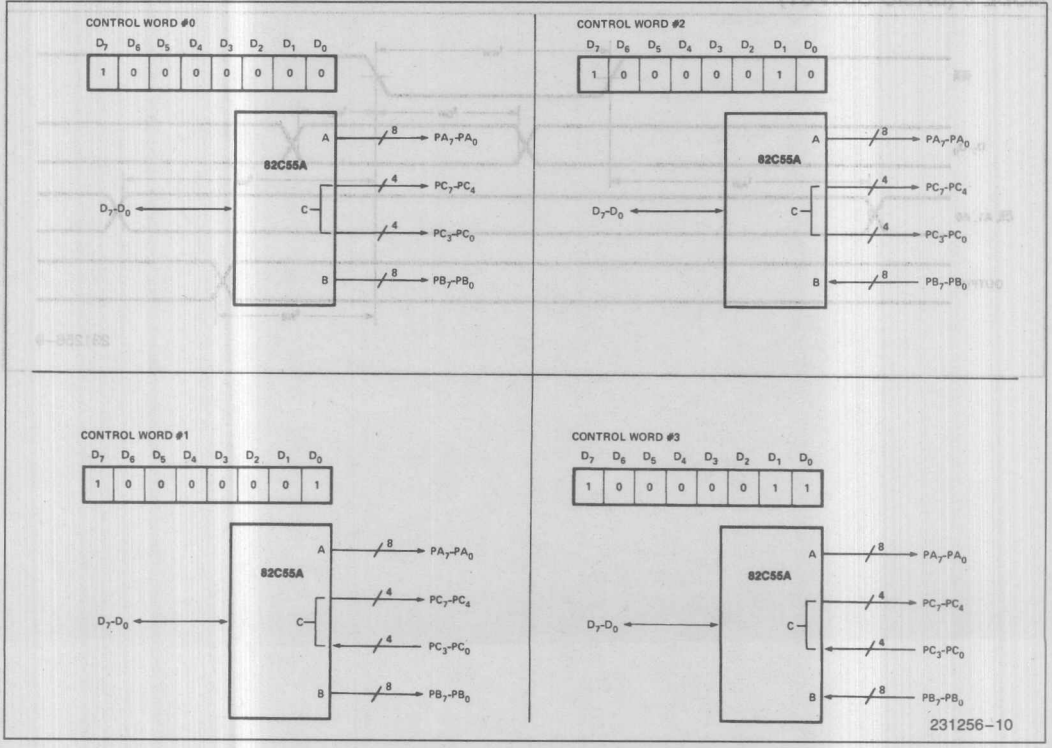
MODE 0 (BASIC OUTPUT)



MODE 0 Port Definition

A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

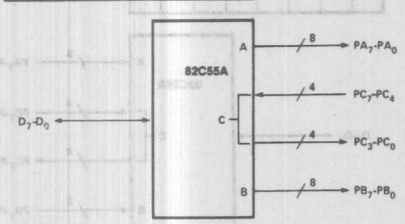
MODE 0 Configurations



MODE 0 Configurations (Continued)

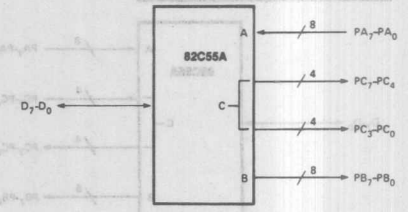
CONTROL WORD #4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



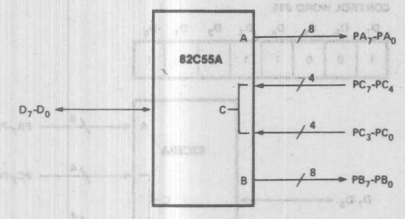
CONTROL WORD #8

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



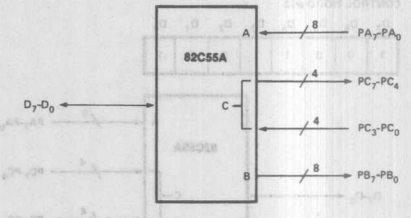
CONTROL WORD #5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



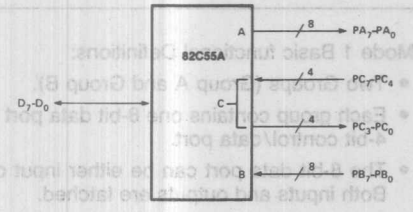
CONTROL WORD #9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



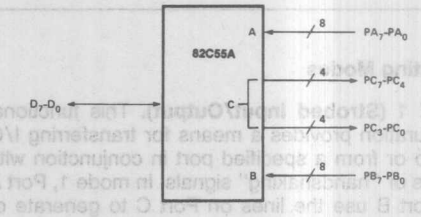
CONTROL WORD #6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



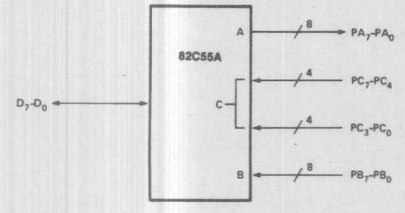
CONTROL WORD #10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



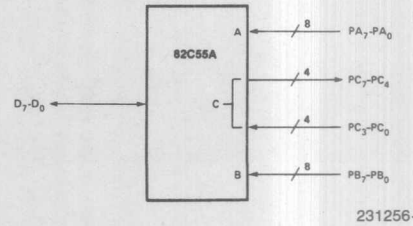
CONTROL WORD #7

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1



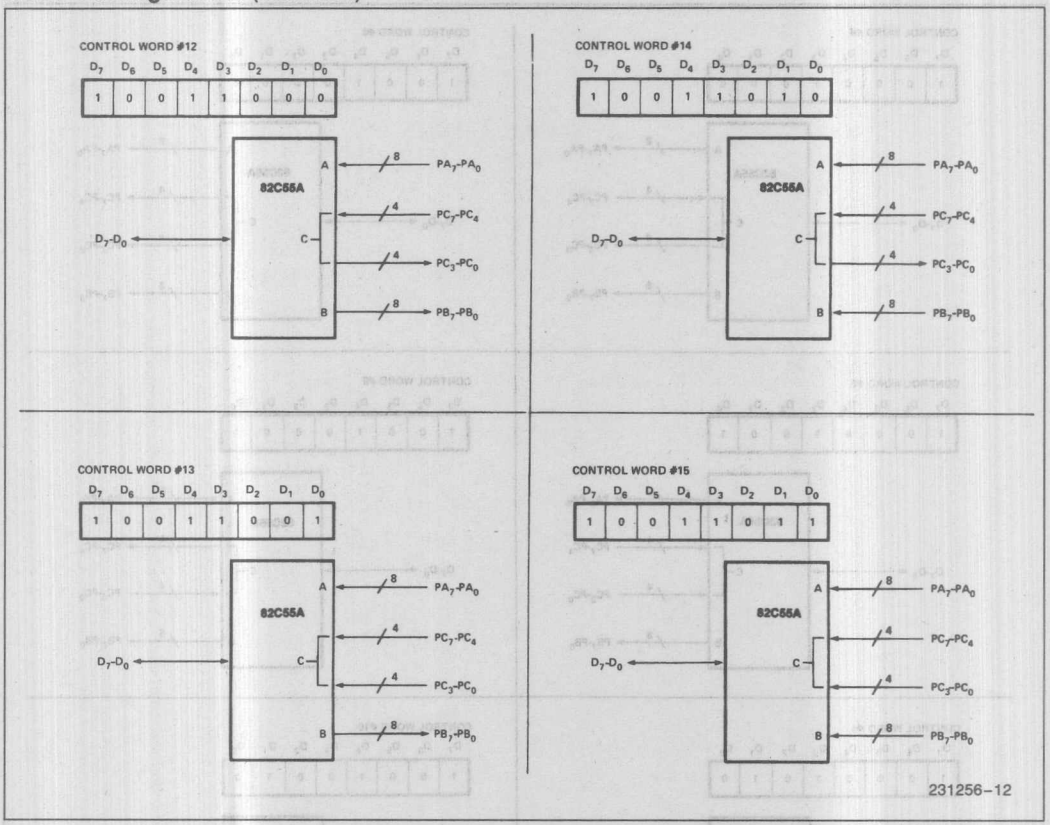
CONTROL WORD #11

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1



231256-11

MODE 0 Configurations (Continued)



Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, Port A and Port B use the lines on Port C to generate or accept these "handshaking" signals.

Mode 1 Basic functional Definitions:

- Two Groups (Group A and Group B).
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by $\overline{STB}$ input being low and is reset by the rising edge of the $\overline{RD}$ input.

INTR (Interrupt Request)

A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the $\overline{STB}$ is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.

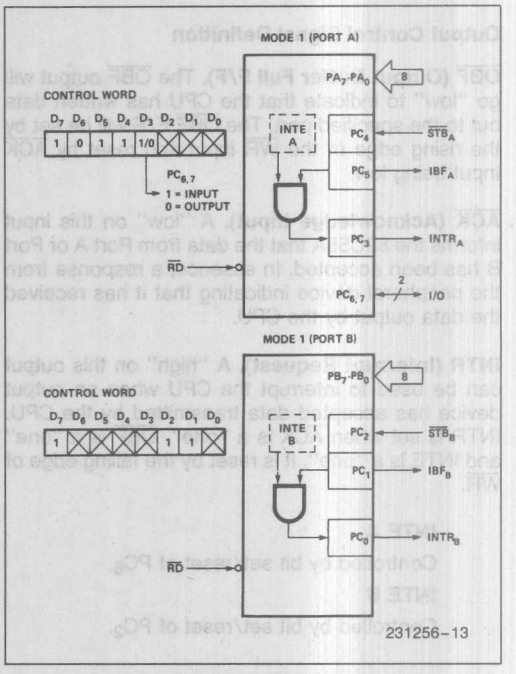


Figure 8. MODE 1 Input

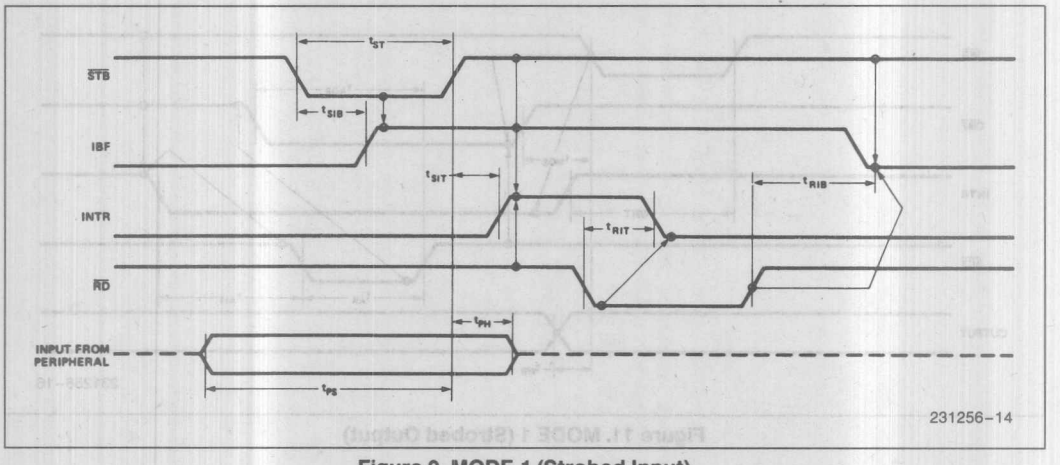


Figure 9. MODE 1 (Strobed Input)

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to the specified port. The $\overline{OBF}$ F/F will be set by the rising edge of the $\overline{WR}$ input and reset by $\overline{ACK}$ Input being low.

$\overline{ACK}$ (Acknowledge Input). A "low" on this input informs the 82C55A that the data from Port A or Port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a "one", $\overline{OBF}$ is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{WR}$.

INTE A

Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.

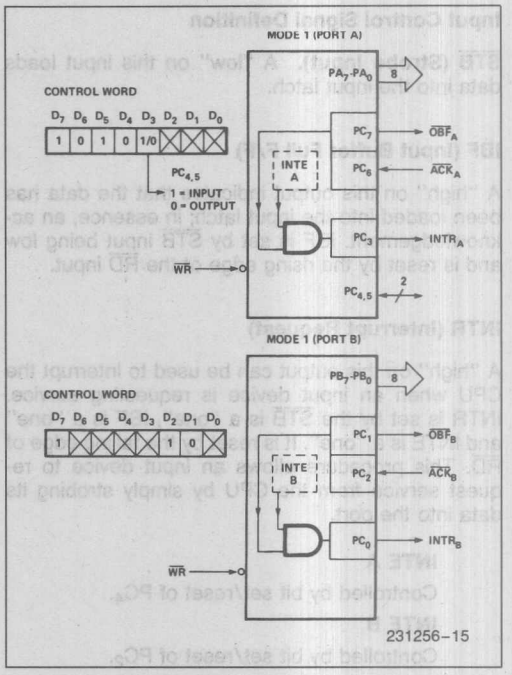


Figure 10. MODE 1 Output

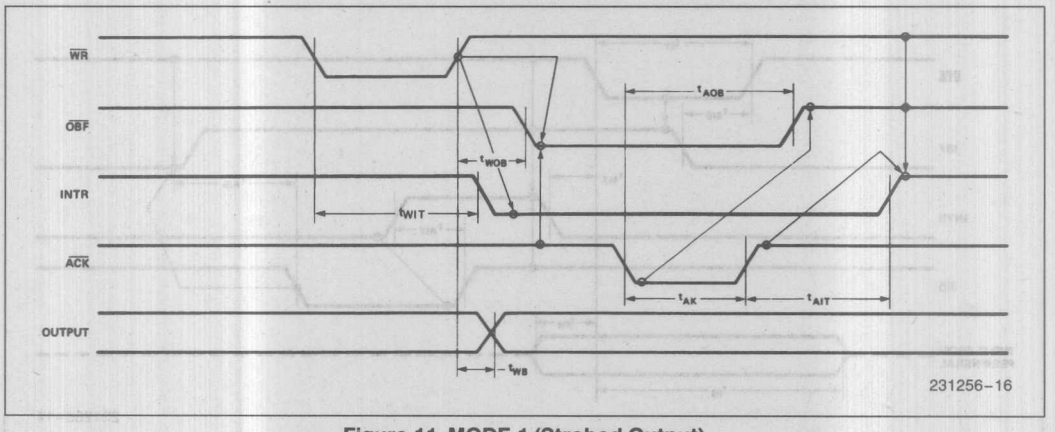


Figure 11. MODE 1 (Strobed Output)

Combinations of MODE 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

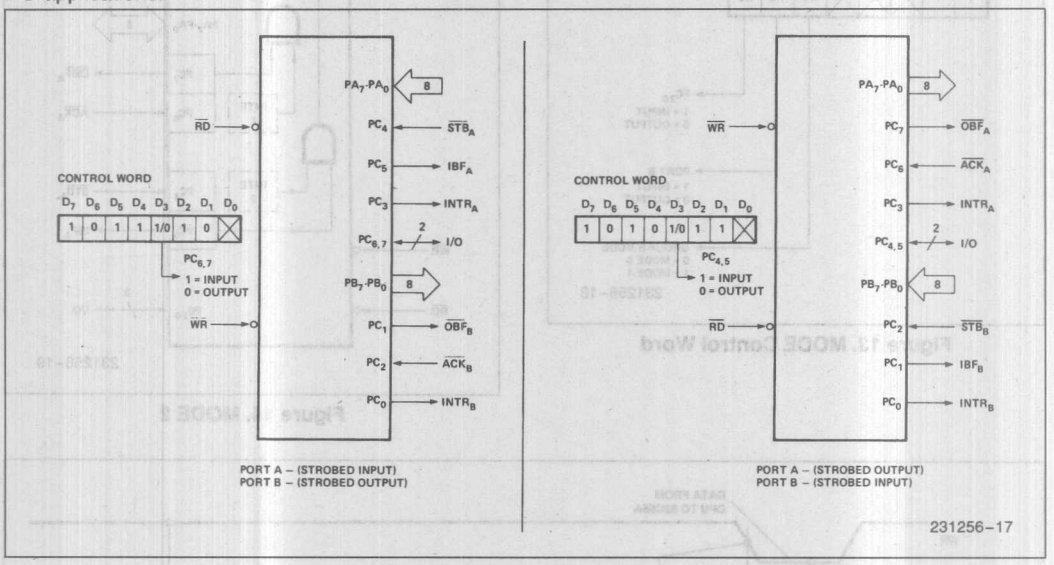


Figure 12. Combinations of MODE 1

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A **only**.
- One 8-bit, bi-directional bus port (Port A) and a 5-bit control port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for input or output operations.

Output Operations

$\overline{OBF}$ (Output Buffer Full). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to port A.

$\overline{ACK}$ (Acknowledge). A "low" on this input enables the tri-state output buffer of Port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (The INTE Flip-Flop Associated with $\overline{OBF}$). Controlled by bit set/reset of PC_6.

Input Operations

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

INTE 2 (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC_4.

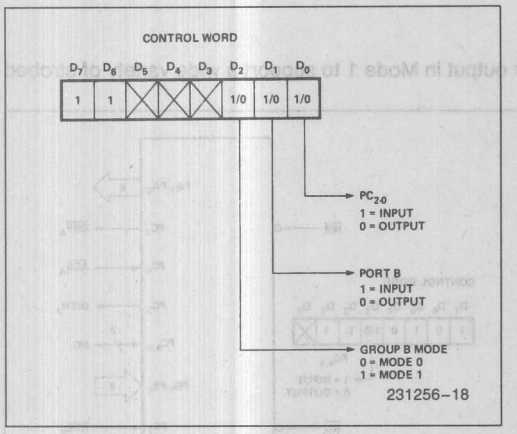


Figure 13. MODE Control Word

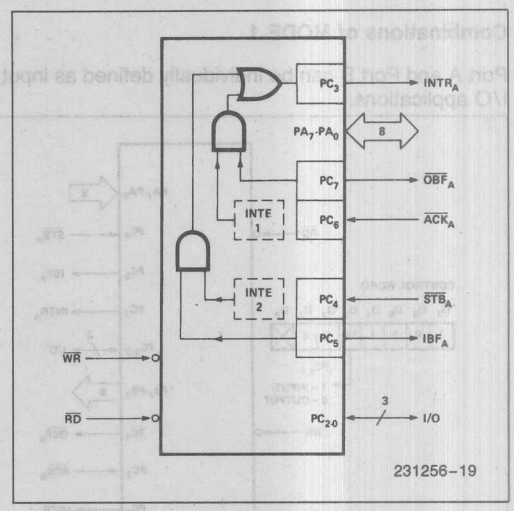


Figure 14. MODE 2

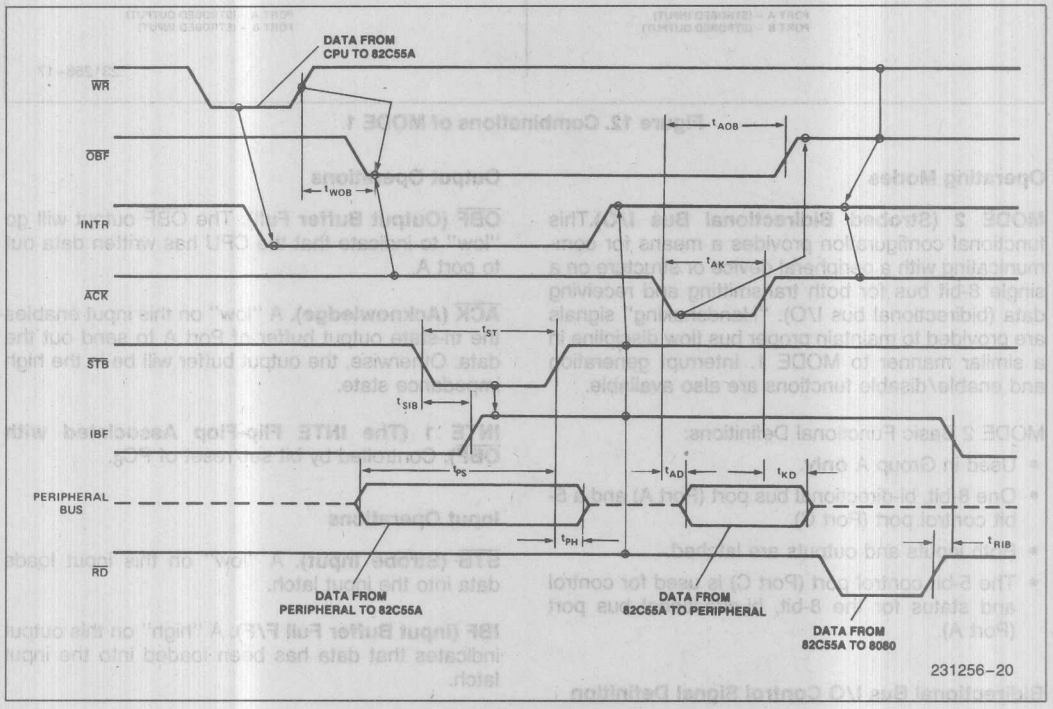


Figure 15. MODE 2 (Bidirectional)

NOTE:
Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$, and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
($INTR = IBF \bullet \overline{MASK} \bullet \overline{STB} \bullet \overline{RD} + \overline{OBF} \bullet \overline{MASK} \bullet \overline{ACK} \bullet \overline{WR}$)

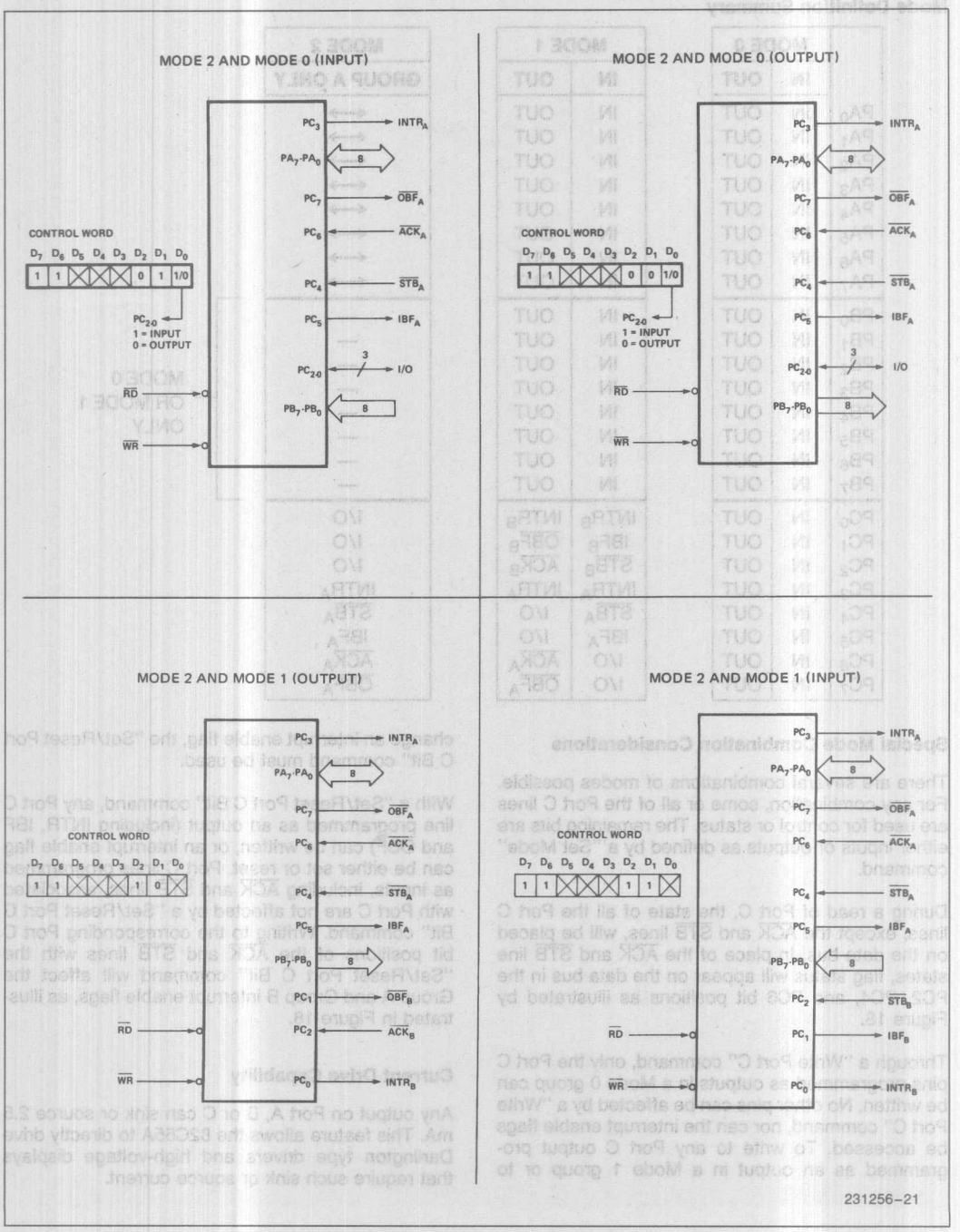


Figure 16. MODE ¼ Combinations

Mode Definition Summary

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	↔	
PA_1	IN	OUT	IN	OUT	↔	
PA_2	IN	OUT	IN	OUT	↔	
PA_3	IN	OUT	IN	OUT	↔	
PA_4	IN	OUT	IN	OUT	↔	
PA_5	IN	OUT	IN	OUT	↔	
PA_6	IN	OUT	IN	OUT	↔	
PA_7	IN	OUT	IN	OUT	↔	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Special Mode Combination Considerations

There are several combinations of modes possible. For any combination, some or all of the Port C lines are used for control or status. The remaining bits are either inputs or outputs as defined by a "Set Mode" command.

During a read of Port C, the state of all the Port C lines, except the $\overline{ACK}$ and $\overline{STB}$ lines, will be placed on the data bus. In place of the $\overline{ACK}$ and $\overline{STB}$ line states, flag status will appear on the data bus in the PC2, PC4, and PC6 bit positions as illustrated by Figure 18.

Through a "Write Port C" command, only the Port C pins programmed as outputs in a Mode 0 group can be written. No other pins can be affected by a "Write Port C" command, nor can the interrupt enable flags be accessed. To write to any Port C output programmed as an output in a Mode 1 group or to change an interrupt enable flag, the "Set/Reset Port C Bit" command must be used.

With a "Set/Reset Port C Bit" command, any Port C line programmed as an output (including INTR, IBF and $\overline{OBF}$) can be written, or an interrupt enable flag can be either set or reset. Port C lines programmed as inputs, including $\overline{ACK}$ and $\overline{STB}$ lines, associated with Port C are not affected by a "Set/Reset Port C Bit" command. Writing to the corresponding Port C bit positions of the $\overline{ACK}$ and $\overline{STB}$ lines with the "Set/Reset Port C Bit" command will affect the Group A and Group B interrupt enable flags, as illustrated in Figure 18.

Current Drive Capability

Any output on Port A, B or C can sink or source 2.5 mA. This feature allows the 82C55A to directly drive Darlington type drivers and high-voltage displays that require such sink or source current.

Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 82C55A is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

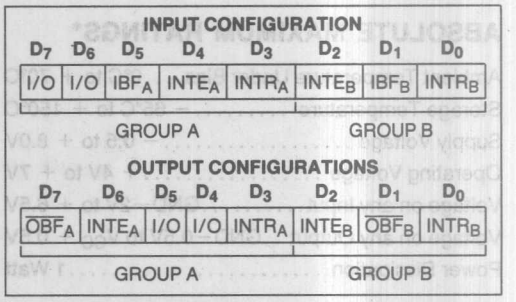


Figure 17a. MODE 1 Status Word Format

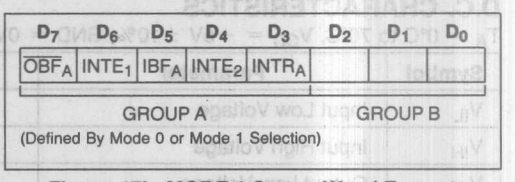


Figure 17b. MODE 2 Status Word Format

Interrupt Enable Flag	Position	Alternate Port C Pin Signal (Mode)
INTE B	PC2	$\overline{ACK}_B$ (Output Mode 1) or $\overline{STB}_B$ (Input Mode 1)
INTE A2	PC4	$\overline{STB}_A$ (Input Mode 1 or Mode 2)
INTE A1	PC6	$\overline{ACK}_A$ (Output Mode 1 or Mode 2

Figure 18. Interrupt Enable Flags in Modes 1 and 2

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to + 70°C
Storage Temperature . . . . . − 65°C to + 150°C
Supply Voltage . . . . . − 0.5 to + 8.0V
Operating Voltage . . . . . + 4V to + 7V
Voltage on any Input . . . . . GND − 2V to + 6.5V
Voltage on any Output . . GND − 0.5V to V_{CC} + 0.5V
Power Dissipation . . . . . 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V (T_A = −40°C to +85°C for Extended Temperture)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC} − 0.4		V V	I_{OH} = −2.5 mA I_{OH} = −100 μA
I_{IL}	Input Leakage Current		±1	μA	V_{IN} = V_{CC} to 0V (Note 1)
I_{OFL}	Output Float Leakage Current		±10	μA	V_{IN} = V_{CC} to 0V (Note 2)
I_{DAR}	Darlington Drive Current	±2.5	(Note 4)	mA	Ports A, B, C R_{ext} = 500Ω V_{ext} = 1.7V
I_{PHL}	Port Hold Low Leakage Current	+50	+300	μA	V_{OUT} = 1.0V Port A only
I_{PHH}	Port Hold High Leakage Current	−50	−300	μA	V_{OUT} = 3.0V Ports A, B, C
I_{PHLO}	Port Hold Low Overdrive Current	−350		μA	V_{OUT} = 0.8V
I_{PHHO}	Port Hold High Overdrive Current	+350		μA	V_{OUT} = 3.0V
I_{CC}	V_{CC} Supply Current		10	mA	(Note 3)
I_{CCSB}	V_{CC} Supply Current-Standby		10	μA	V_{CC} = 5.5V V_{IN} = V_{CC} or GND Port Conditions If I/P = Open/High O/P = Open Only With Data Bus = High/Low $\overline{CS}$ = High Reset = Low Pure Inputs = Low/High

NOTES:
1. Pins A_1, A_0, $\overline{CS}$, $\overline{WR}$, $\overline{RD}$, Reset.
2. Data Bus; Ports B, C.
3. Outputs open.
4. Limit output current to 4.0 mA.

CAPACITANCE

T_A = 25°C, V_{CC} = GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	Unmeasured pins returned to GND f_c = 1 MHz(5)
$C_{I/O}$	I/O Capacitance		20	pF	

NOTE:
5. Sampled not 100% tested.

A.C. CHARACTERISTICS

T_A = 0° to 70°C, V_{CC} = +5V ±10%, GND = 0V

T_A = −40°C to +85°C for Extended Temperature

BUS PARAMETERS

READ CYCLE

Symbol	Parameter	82C55A-2		Units	Test Conditions
		Min	Max		
t_{AR}	Address Stable Before $\overline{RD}$ ↓	0		ns	
t_{RA}	Address Hold Time After $\overline{RD}$ ↑	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	150		ns	
t_{RD}	Data Delay from $\overline{RD}$ ↓		120	ns	
t_{DF}	$\overline{RD}$ ↑ to Data Floating	10	75	ns	
t_{RV}	Recovery Time between $\overline{RD}$/$\overline{WR}$	200		ns	

WRITE CYCLE

Symbol	Parameter	82C55A-2		Units	Test Conditions
		Min	Max		
t_{AW}	Address Stable Before $\overline{WR}$ ↓	0		ns	
t_{WA}	Address Hold Time After $\overline{WR}$ ↑	20		ns	Ports A & B
		20		ns	Port C
t_{WW}	$\overline{WR}$ Pulse Width	100		ns	
t_{DW}	Data Setup Time Before $\overline{WR}$ ↑	100		ns	
t_{WD}	Data Hold Time After $\overline{WR}$ ↑	30		ns	Ports A & B
		30		ns	Port C

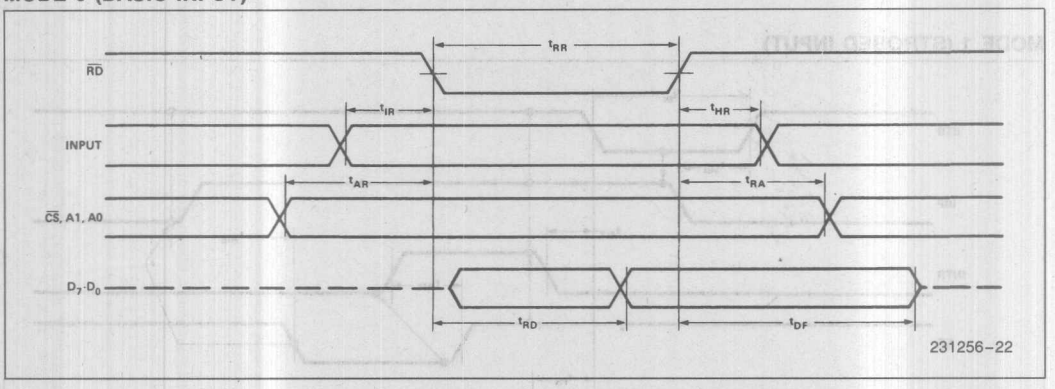
OTHER TIMINGS

Symbol	Parameter	82C55A-2		Units Conditions	Test
		Min	Max		
t_{WB}	$\overline{WR}$ = 1 to Output		350	ns	
t_{IR}	Peripheral Data Before $\overline{RD}$	0		ns	
t_{HR}	Peripheral Data After $\overline{RD}$	0		ns	
t_{AK}	$\overline{ACK}$ Pulse Width	200		ns	
t_{ST}	$\overline{STB}$ Pulse Width	100		ns	
t_{PS}	Per. Data Before $\overline{STB}$ High	20		ns	
t_{PH}	Per. Data After $\overline{STB}$High	50		ns	
t_{AD}	$\overline{ACK}$ = 0 to Output		175	ns	
t_{KD}	$\overline{ACK}$ = 1 to Output Float	20	250	ns	
t_{WOB}	$\overline{WR}$ = 1 to $\overline{OBF}$ = O		150	ns	
t_{AOB}	$\overline{ACK}$ = 0 to $\overline{OBF}$ = 1		150	ns	
t_{SIB}	$\overline{STB}$ = 0 to IBF = 1		150	ns	
t_{RIB}	$\overline{RD}$ = 1 to IBF = 0		150	ns	
t_{RIT}	$\overline{RD}$ = 0 to INTR = 0		200	ns	
t_{SIT}	$\overline{STB}$ = 1 to INTR = 1		150	ns	
t_{AIT}	$\overline{ACK}$ = 1 to INTR = 1		150	ns	
t_{WIT}	$\overline{WR}$ = 0 to INTR = 0		200	ns	see note 1
t_{RES}	Reset Pulse Width	500		ns	see note 2

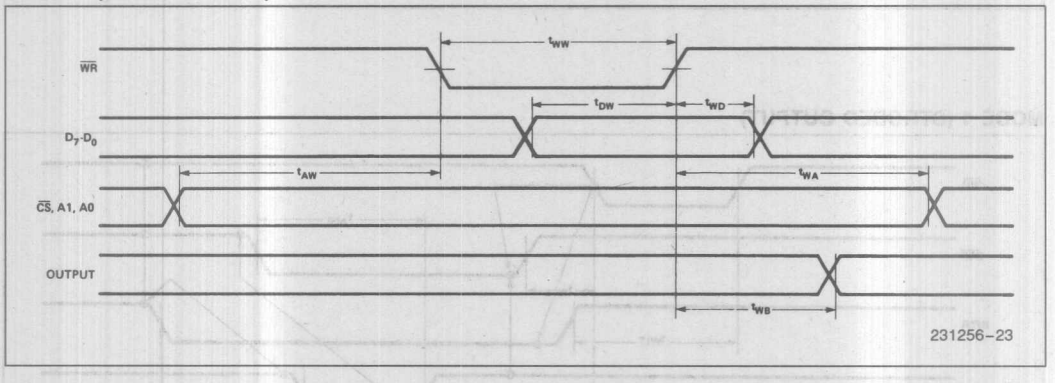
NOTE:
1. INTR ↑ may occur as early as $\overline{WR}$ ↓.
2. Pulse width of initial Reset pulse after power on must be at least 50 μSec. Subsequent Reset pulses may be 500 ns minimum.

WAVEFORMS

MODE 0 (BASIC INPUT)

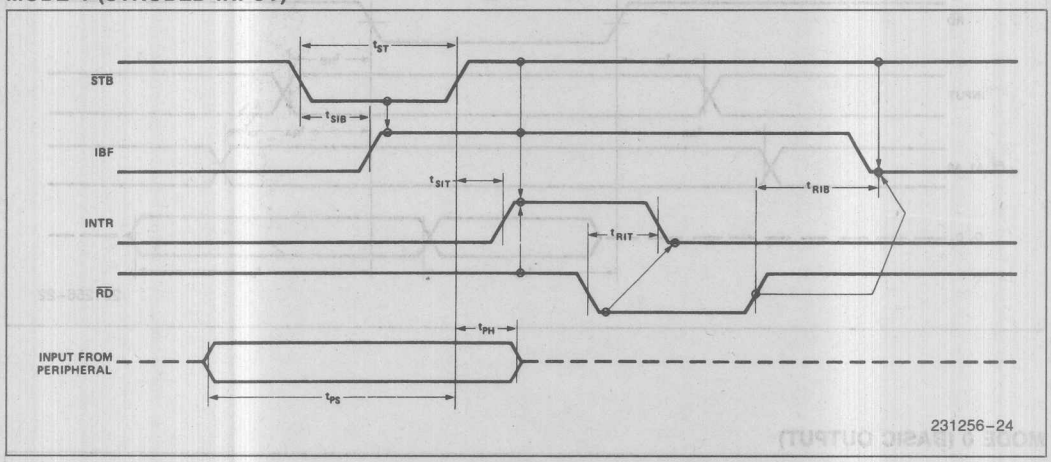


MODE 0 (BASIC OUTPUT)

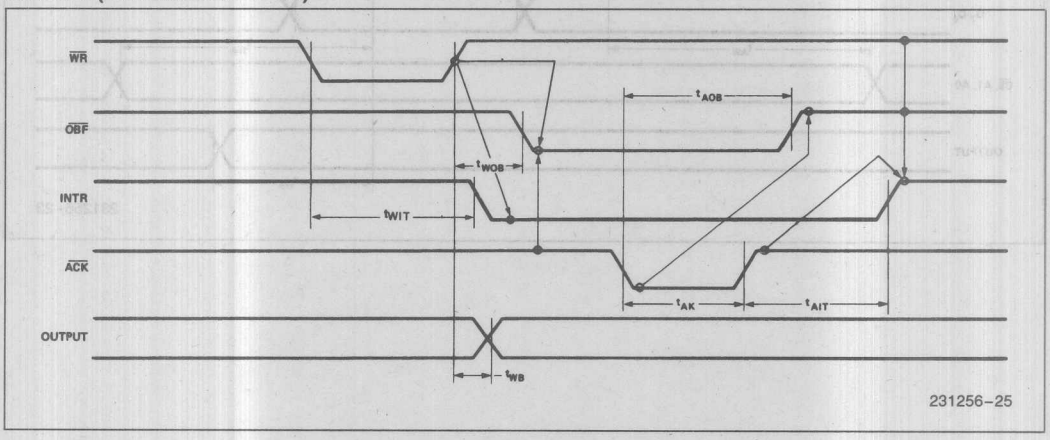


WAVEFORMS (Continued)

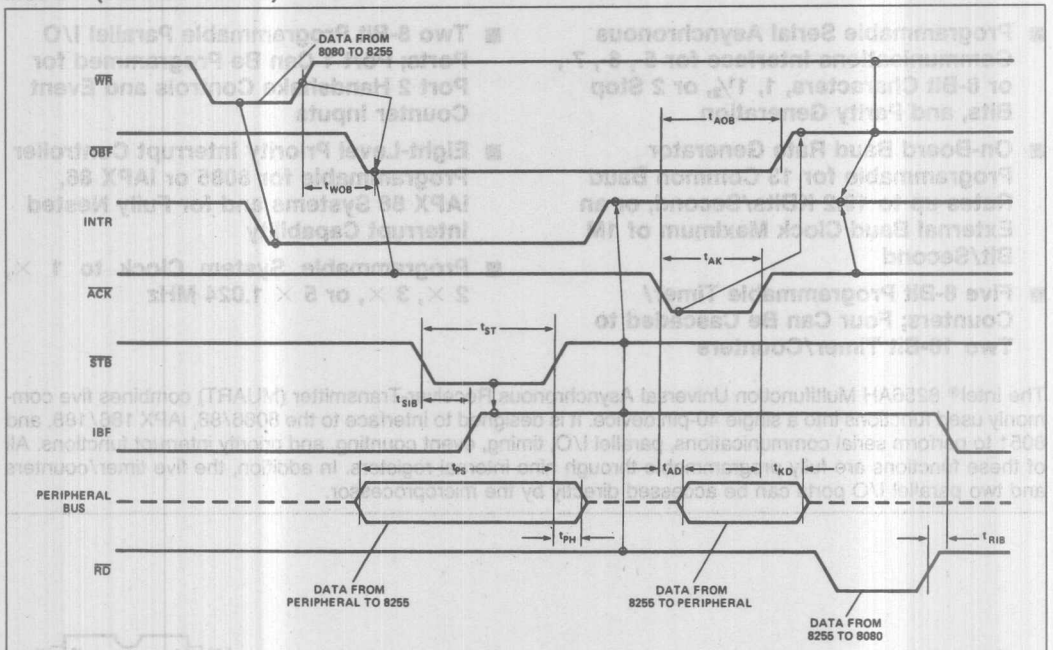
MODE 1 (STROBED INPUT)



MODE 1 (STROBED OUTPUT)



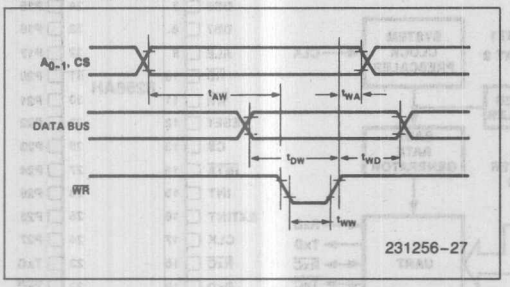
MODE 2 (BIDIRECTIONAL)



231256-26

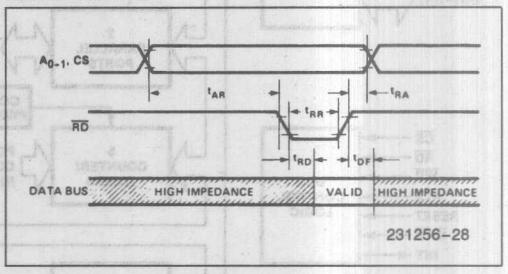
Note:
Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ AND $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
($INTR = IBF \bullet \overline{MASK} \bullet \overline{STB} \bullet \overline{RD} + \overline{OBF} \bullet \overline{MASK} \bullet \overline{ACK} \bullet \overline{WR}$)

WRITE TIMING



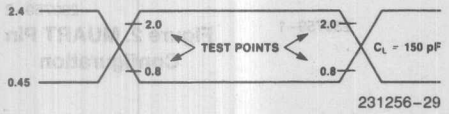
231256-27

READ TIMING



231256-28

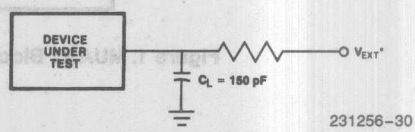
A.C. TESTING INPUT, OUTPUT WAVEFORM



231256-29

A.C. Testing Inputs Are Driven At 2.4V For A Logic 1 And 0.45V For A Logic 0 Timing Measurements Are Made At 2.0V For A Logic 1 And 0.8 For A Logic 0.

A.C. TESTING LOAD CIRCUIT



231256-30

*V_{EXT} Is Set At Various Voltages During Testing To Guarantee The Specification. C_L Includes Jig Capacitance.

8256AH
MULTIFUNCTION MICROPROCESSOR SUPPORT CONTROLLER

- **Programmable Serial Asynchronous Communications Interface for 5-, 6-, 7-, or 8-Bit Characters, 1, 1½, or 2 Stop Bits, and Parity Generation**
- **On-Board Baud Rate Generator Programmable for 13 Common Baud Rates up to 19.2 KBits/Second, or an External Baud Clock Maximum of 1M Bit/Second**
- **Five 8-Bit Programmable Timer/Counters; Four Can Be Cascaded to Two 16-Bit Timer/Counters**
- **Two 8-Bit Programmable Parallel I/O Ports; Port 1 Can Be Programmed for Port 2 Handshake Controls and Event Counter Inputs**
- **Eight-Level Priority Interrupt Controller Programmable for 8085 or iAPX 86, iAPX 88 Systems and for Fully Nested Interrupt Capability**
- **Programmable System Clock to 1 ×, 2 ×, 3 ×, or 5 × 1.024 MHz**

The Intel® 8256AH Multifunction Universal Asynchronous Receiver-Transmitter (MUART) combines five commonly used functions into a single 40-pin device. It is designed to interface to the 8086/88, iAPX 186/188, and 8051 to perform serial communications, parallel I/O, timing, event counting, and priority interrupt functions. All of these functions are fully programmable through nine internal registers. In addition, the five timer/counters and two parallel I/O ports can be accessed directly by the microprocessor.

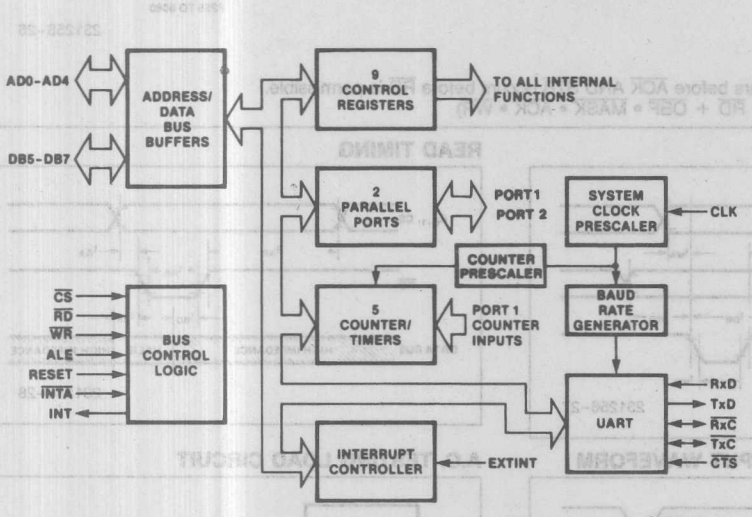


Figure 1. MUART Block Diagram

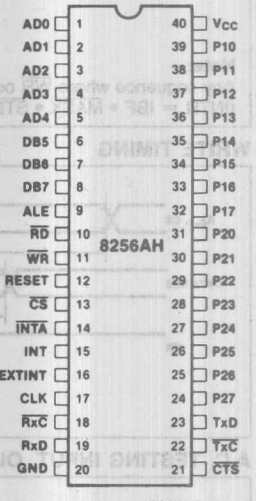


Figure 2. MUART Pin Configuration

September 1987
Order Number: 230759-002

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
AD0–AD4 DB5–DB7	1–5 6–8	I/O	**ADDRESS/DATA:** Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, AD0–AD3 are used to select the proper register, while AD1–AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.
ALE	9	I	**ADDRESS LATCH ENABLE:** Latches the 5 address lines on AD0–AD4 and $\overline{CS}$ on the falling edge.
$\overline{RD}$	10	I	**READ CONTROL:** When this signal is low, the selected register is gated onto the data bus.
$\overline{WR}$	11	I	**WRITE CONTROL:** When this signal is low, the value on the data bus is written into the selected register.
RESET	12	I	**RESET:** An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.
$\overline{CS}$	13	I	**CHIP SELECT:** A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and $\overline{RD}$ and $\overline{WR}$ have no effect unless $\overline{CS}$ was latched low during the ALE cycle.
$\overline{INTA}$	14	I	**INTERRUPT ACKNOWLEDGE:** If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.
INT	15	O	**INTERRUPT REQUEST:** A high signals the microprocessor that the MUART needs service.
EXTINT	16	I	**EXTERNAL INTERRUPT:** An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an $\overline{INTA}$ occurs or the interrupt address register is read.
CLK	17	I	**SYSTEM CLOCK:** The reference clock for the baud rate generator and the timers.
RxC	18	I/O	**RECEIVE CLOCK:** If the baud rate bits in the Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1–0FH, this pin outputs a square wave whose rising edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.
RxD	19	I	**RECEIVE DATA:** Serial data input.
GND	20	PS	**GROUND:** Power supply and logic ground reference.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
$\overline{CTS}$	21	I	**CLEAR TO SEND:** This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected $\overline{CTS}$ is level sensitive. As long as $\overline{CTS}$ is low, any character loaded into the transmitter buffer register will be transmitted serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1–0FH is selected, $\overline{CTS}$ must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where 1/2 the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If $\overline{CTS}$ is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on $\overline{CTS}$ occurs. If 0.75 stop bits is chosen, the $\overline{CTS}$ input is edge sensitive. A negative edge on $\overline{CTS}$ results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on $\overline{CTS}$. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.
TxC	22	I/O	**TRANSMIT CLOCK:** If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32× or 64× clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3–0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1 1/2 or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.
TxD	23	O	**TRANSMIT DATA:** Serial data output.
P27–P20	24–31	I/O	**PARALLEL I/O PORT 2:** Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.
P17–P10	32–39	I/O	**PARALLEL I/O PORT 1:** Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.
V_{CC}	40	PS	**POWER:** +5V power supply.

FUNCTIONAL DESCRIPTION

The 8256AH Multi-Function Universal Asynchronous Receiver-Transmitter (MUART) combines five commonly used functions into a single 40-pin device. The MUART performs asynchronous serial communications, parallel I/O, timing, event counting, and interrupt control. For detailed application information, see Intel AP Note #153, Designing with the 8256.

Serial Communications

The serial communications portion of the MUART contains a full-duplex asynchronous receiver-transmitter (UART). A programmable baud rate generator is included on the MUART to permit a variety of operating speeds without external components. The UART can be programmed by the CPU for a variety of character sizes, parity generation and detection, error detection, and start/stop bit handling. The receiver checks the start and stop bits in the center of the bit, and a break halts the reception of data. The transmitter can send breaks and can be controlled by an external enable pin.

Parallel I/O

The MUART includes 16 bits of general purpose parallel I/O. Eight bits (Port 1) can be individually changed from input to output or used for special I/O functions. The other eight bits (Port 2) can be used as nibbles (4 bits) or as bytes. These eight bits also include a handshaking capability using two pins on Port 1.

Counter/Timers

There are five 8-bit counter/timers on the MUART. The timers can be programmed to use either a 1 kHz or 16 kHz clock generated from the system clock. Four of the 8-bit counter/timers can be cascaded to two 16-bit counter/timers, and one of the 8-bit counter/timers can be reset to its initial value by an external signal.

Interrupts

An eight-level priority interrupt controller can be configured for fully nested or normal interrupt priority. Seven of the eight interrupts service functions on the MUART (counter/timers, UART), and one external interrupt is provided which can be used for a particular function or for chaining interrupt controllers or more MUARTs. The MUART will support 8085 and 8086/88 systems with direct interrupt vectoring, or the MUART can be polled to determine the cause of the interrupt. If additional interrupt control capability is needed, the MUART's interrupt controller can be cascaded into another MUART, into an Intel 8259A Programmable Interrupt Controller, or into the interrupt controller of the iAPX 186/188 High-Integration Microprocessor.

INITIALIZATION

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, **Command Byte 1** must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

Command byte 1
Command byte 2
Command byte 3
Mode byte
Port 1 control
Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. The MUART should be reset before initialization. (Either a hardware or a software reset will do.)

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 80186 microprocessor. Figure 3 displays the block diagram for this interface. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0–AD3 are the address lines. For 16-bit microprocessors, AD1–AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed ad-

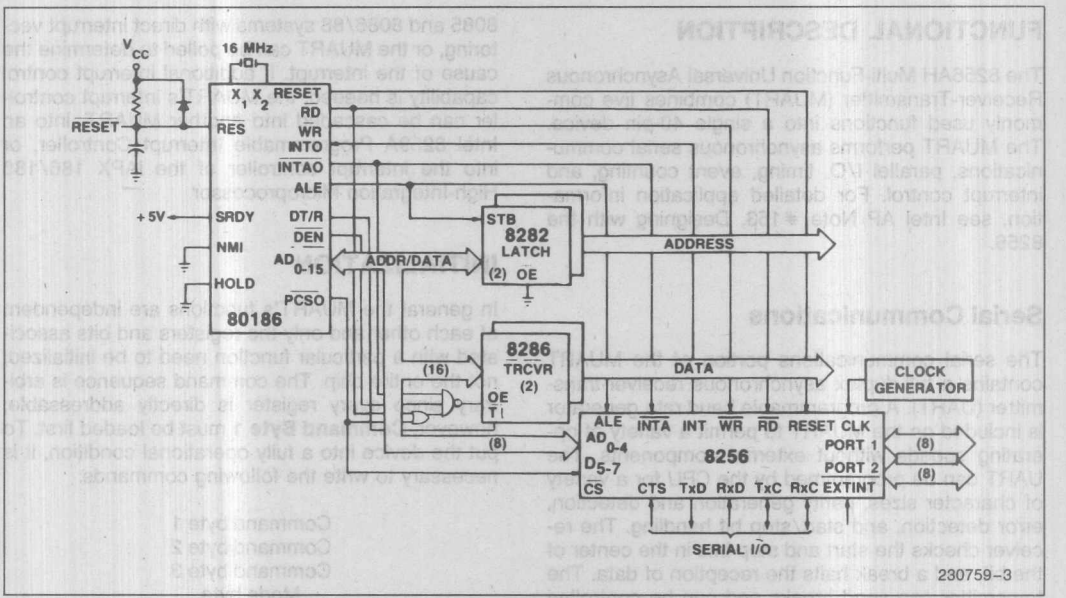


Figure 3. 80186/8256 Interface

dress/data lines or multiplexed address/status lines. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte, the internal registers will be 512 address locations apart and the chip would occupy an 8k word address space.

DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 2 lists the registers and their addresses.

Command Register 1

L1	L0	S1	S0	BRKI	BITI	8086	FRQ
(0R)				(0W)			

FRQ—TIMER FREQUENCY SELECT

This bit selects between two frequencies for the five timers. If FRQ = 0, the timer input frequency is 16 kHz (62.5 μs). If FRQ = 1, the timer input frequency is 1 kHz (1 ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

8086—8086 MODE ENABLE

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086 = 0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first $\overline{INTA}$. In 8086 mode (8086 = 1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to $\overline{INTA}$ is for 8086 interrupts where the first $\overline{INTA}$ is ignored, and an interrupt vector (40H to 47H) is placed on the bus in response to the second $\overline{INTA}$.

BITI—INTERRUPT ON BIT CHANGE

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

Table 2. MUART Registers

Read Registers								8085 Mode: AD3 AD2 AD1 AD0 8086 Mode: AD4 AD3 AD2 AD1				Write Registers							
L1	L0	S1	S0	BRKI	BITI	8086	FRQ	0	0	0	0	L1	L0	S1	S0	BRKI	BITI	8086	FRQ
Command 1												Command 1							
PEN	EP	C1	C0	B3	B2	B1	B0	0	0	0	1	PEN	EP	C1	C0	B3	B2	B1	B0
Command 2												Command 2							
0	RxE	IAE	NIE	0	SBRK	TBRK	0	0	0	1	0	SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
Command 3												Command 3							
T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0	0	0	1	1	T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
Mode												Mode							
P17	P16	P15	P14	P13	P12	P11	P10	0	1	0	0	P17	P16	P15	P14	P13	P12	P11	P10
Port 1 Control												Port 1 Control							
L7	L6	L5	L4	L3	L2	L1	L0	0	1	0	1	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Enable												Set Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	0	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Address												Reset Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Receiver Buffer												Transmitter Buffer							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Port 1												Port 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Port 2												Port 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 1												Timer 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 2												Timer 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 3												Timer 3							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 4												Timer 4							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 5												Timer 5							
INT	REF	TBE	TRE	BD	PE	OE	FE	1	1	1	1	0	RS4	RS3	RS2	RS1	RS0	TME	DSC
Status												Modification							

BRKI—BREAK-IN DETECT ENABLE

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In. A Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1—STOP BIT LENGTH

S1	S0	Stop Bit Length
0	0	1
0	1	1.5
1	0	2
1	1	0.75

The relationship of the number of stop bits and the function of input $\overline{CTS}$ is discussed in the Pin Description section under "$\overline{CTS}$".

L0, L1—CHARACTER LENGTH

L1	L0	Character Length
0	0	8
0	1	7
1	0	6
1	1	5

Command Register 2

PEN	EP	C1	C0	B3	B2	B1	B0
(1R)				(1W)			

Programming bits 0 . . . 3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0 . . . 3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32× or 64× the baud rate. The data transmission rates range from 0. . .32 Kbaud.

If bits 0 . . . 3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 MHz to 1.024 MHz.

B0, B1, B2, B3—BAUD RATE SELECT

These four bits select the bit clock's source, sampling rate, and serial rate for the internal baud rate generator.

B3	B2	B1	B0	Baud Rate	Sampling Rate
0	0	0	0	$\overline{TxC}$, $\overline{RxC}$	1
0	0	0	1	$\overline{TxC}$/64	64
0	0	1	0	$\overline{TxC}$/32	32
0	0	1	1	19200	32
0	1	0	0	9600	64
0	1	0	1	4800	64
0	1	1	0	2400	64
0	1	1	1	1200	64
1	0	0	0	600	64
1	0	0	1	300	64
1	0	1	0	200	64
1	0	1	1	150	64
1	1	0	0	110	64
1	1	0	1	100	64
1	1	1	0	75	64
1	1	1	1	50	64

The following table gives an overview of the function of pins TxC and RxC:

Bits 3 to 0 (Hex.)	TxC	RxC
0	Input: 1 × baud rate clock for the transmitter	Input: 1 × baud rate clock for the receiver
1, 2	Input: 32 × or 64 × baud rate for transmitter and receiver	Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level
3 to F	Output: baud rate clock of the transmitter	Output: as above

As an output, RxC outputs a low-to-high transition at sampling time of every data bit of a character. Thus, data can be loaded, e.g., into a shift register externally. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1—SYSTEM CLOCK PRESCALER (BITS 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP—EVEN PARITY (BIT 6)

EP = 0: Odd parity
EP = 1: Even parity

PEN—PARITY ENABLE (BIT 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Even parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIW	END	SBRK	TBRK	RST
	(2R)				(2W)		

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0–6 is low, no change occurs to that bit. When command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST—RESET

If RST is set, the following events occur:

1. All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.
2. The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.
3. The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.
4. If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does *not* alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK—TRANSMIT BREAK

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of $\overline{CTS}$ is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

SBRK—SINGLE CHARACTER BREAK

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle (marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END—END OF INTERRUPT

If fully nested interrupt mode is selected, this bit resets the currently served interrupt level in the Interrupt Service Register. *This command must occur at the end of each interrupt service routine during fully nested interrupt mode.* END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE—NESTED INTERRUPT ENABLE

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section of AP-153 under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE—INTERRUPT ACKNOWLEDGE ENABLE

This bit enables an automatic response to $\overline{\text{INTA}}$. The particular response is determined by the 8086 bit in Command Register 1.

RxE—RECEIVE ENABLE

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET—BIT SET/RESET

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

MODE REGISTER

T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
(3R)					(3W)		

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3–B0 with a value ≥ 3H.

P2C2, P2C1, P2C0—PORT 2 CONTROL

P2C2	P2C1	P2C0	Mode	Direction	
				Upper	Lower
0	0	0	Nibble	Input	Input
0	0	1	Nibble	Input	Output
0	1	0	Nibble	Output	Input
0	1	1	Nibble	Output	Output
1	0	0	Byte Handshake	Input	
1	0	1	Byte Handshake	Output	
1	1	0	*DO NOT USE*		
1	1	1	Test		

NOTE:
If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3—COUNTER/TIMER MODE

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C—TIMER 5 CONTROL

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues

counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

T35, T24—CASCADE TIMERS

These two bits cascade Timers 3 and 5 or 2 and 4.

Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. It T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse.

When a high-to-low transition occurs, Timer 5 is preset to its saved value, but Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 3.

NOTE:
Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

Port 1 Control Register

P17	P16	P15	P14	P13	P12	P11	P10
(4R)					(4W)		

Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it is low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

Port 10, 11—HANDSHAKE CONTROL

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as $\overline{STB}/\overline{ACK}$ handshake control input, and Port 11 is programmed as $\overline{IBF}/\overline{OBF}$ handshake control output.

If byte handshake mode is enabled for output on Port 2 $\overline{OBF}$ indicates that a character has been loaded into the Port 2 output buffer. When an external

Table 3. Event Counters/Timers Mode of Operation

Event Counter/ Timer	Function	Programming (Mode Word)	Clock Source
1	8-bit timer	—	Internal clock
2	8-bit timer	T24 = 0, CT2 = 0	Internal clock
	8-bit event counter	T24 = 0, CT2 = 1	P12 pin 37
2	8-bit timer	T35 = 0, CT3 = 0	Internal clock
	8-bit event counter	T35 = 0, CT3 = 1	P13 pin 36
4	8-bit timer	T24 = 0	Internal clock
5	8-bit timer, normal mode	T35 = 0, T5C = 0	Internal clock
	8-bit timer, retriggerable mode	T35 = 0, T5C = 1	Internal clock
2 and 4 cascaded	16-bit timer	T24 = 1, CT2 = 0	Internal clock
	16-bit event counter	T24 = 1, CT2 = 1	P12 pin 37
3 and 5 cascaded	16-bit timer, normal mode	T35 = 1, T5C = 0, CT3 = 0	Internal clock
	16-bit event counter, normal mode	T35 = 1, T5C = 0, CT3 = 1	P13 pin 36
	16-bit timer, retriggerable mode	T35 = 1, T5C = 1, CT3 = 0	Internal clock
	16-bit event counter, retriggerable mode	T35 = 1, T5C = 1, CT3 = 1	P13 pin 36

device reads the data, it acknowledges this operation by driving $\overline{ACK}$ low. $\overline{OBF}$ is set low by writing to Port 2 and is reset by $\overline{ACK}$.

If byte handshake mode is enabled for input on Port 2, $\overline{STB}$ is an input. $\overline{IBF}$ is driven low after $\overline{STB}$ goes low. On the rising edge of $\overline{STB}$ the data from Port 2 is latched.

$\overline{IBF}$ is reset high when Port 2 is read.

PORT 12, 13—COUNTER 2, 3 INPUT

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

PORT 14—BAUD RATE GENERATOR OUTPUT CLOCK

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 × the serial bit rate excet at 19.2 Kbps when it is 32 × the bit rate.

PORT 15—TIMER 5 TRIGGER

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the slave register and starts the timer.

PORT 16—BREAK-IN DETECT

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

PORT 17—PORT INTERRUPT SOURCE

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

Interrupt Enable Register

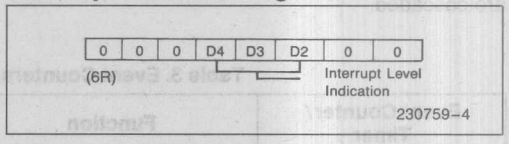
L7	L6	L5	L4	L3	L2	L1	L0

(5R) (5W = enable, 6W = disable)

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt Enable Register (5R).

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receive Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

Interrupt Address Register



Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge $\overline{INTA}$; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer

D7	D6	D5	D4	D3	D2	D1	D0

(7R) (7W)

Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set.

Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming $\overline{CTS}$ is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

D7	D6	D5	D4	D3	D2	D1	D0
(8R)					(8W)		

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

D7	D6	D5	D4	D3	D2	D1	D0
(9R)					(9W)		

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1–5

D7	D6	D5	D4	D3	D2	D1	D0
($0A_{16}$–OE_{16}R)					($0A_{16}$–OE_{16}W)		

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while $\overline{RD}$ is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will unlatch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded, writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X *256 + 255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

INT	RBF	TBE	TRE	BD	PE	OE	FE

($0F_{16}$R)

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt in INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE—FRAMING ERROR, TRANSMISSION MODE

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the *first* stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is required to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

OE—OVERRUN ERROR

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

PE—PARITY ERROR

This bit indicates a parity error has occurred during the reception of a character. A parity error is present

if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD—BREAK/BREAK-IN

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section of AP-153 under "Receive Break Detect" and "Break-In Detect."

TRE—TRANSMIT REGISTER EMPTY

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If $\overline{CTS}$ is low, the Transmitter Register will be loaded during the transmission of the start bit. If $\overline{CTS}$ is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until $\overline{CTS}$ goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE—TRANSMITTER BUFFER EMPTY

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF—RECEIVER BUFFER FULL

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT—INTERRUPT PENDING

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by $\overline{INTA}$ or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

MODIFICATION REGISTER

0	RS4	RS3	RS2	RS1	RS0	TME	DSC

($0F_{16}W$)

DSC—DISABLE START BIT CHECK

DSC disables the receivers start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME—TRANSMISSION MODE ENABLE

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the status register.

RS0, RS1, RS2, RS3, RS4—RECEIVER SAMPLE TIME

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is *not* clocked by RxC.

NOTE:
The modification register cannot be read. Reading from address 0FH, 8086: 1EH gates the contents of the status register onto the data bus.

A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:

- The start bit check is enabled.
- Status Register Bit 0 (FE) indicates framing error.
- The sampling time of the serial receiver is the bit center.

A software reset (Command Word 3, RST) does not affect the modification register.

Hardware Reset

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

1) Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be intputs and event counters/timers are configured as independent 8-bit timers.
2) Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.
3) The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT IS INACTIVE (LOW).
4) The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.
5) The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.
6) Status Register Bit 0 implies framing error.
7) The receiver samples input RxD at bit center.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

RS4	RS3	RS2	RS1	RS0	Point of Time Between Start of Bit and End of Bit Measured in Steps of 1/32 Bit Length
0	1	1	1	1	1 (Start of Bit)
0	1	1	1	0	2
0	1	1	0	1	3
0	1	1	0	0	4
0	1	0	1	1	5
0	1	0	1	0	6
0	1	0	0	1	7
0	1	0	0	0	8
0	0	1	1	1	9
0	0	1	1	0	10
0	0	1	0	1	11
0	0	1	0	0	12
0	0	0	1	1	13
0	0	0	1	0	14
0	0	0	0	1	15
0	0	0	0	0	16 (Bit center)
1	1	1	1	1	17
1	1	1	1	0	18
1	1	1	0	1	19
1	1	1	0	0	20
1	1	0	1	1	21
1	1	0	1	0	22
1	1	0	0	1	23
1	1	0	0	0	24
1	0	1	1	1	25
1	0	1	1	0	26
1	0	1	0	1	27
1	0	1	0	0	28
1	0	0	1	1	29
1	0	0	1	0	30
1	0	0	0	1	31
1	0	0	0	0	32 (End of Bit)

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On Any Pin
with Respect to Ground.....−0.5V to +7V
Power Dissipation 1W

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5.0V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Leakage		10 −10	μA μA	$V_{IN} = V_{CC}$ V_{IN} = 0V
I_{LO}	Output Leakage		10 −10	μA μA	$V_{OUT} = V_{CC}$ V_{OUT} = 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz(1)
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured Pins Returned to V_{SS}(1)

NOTE:
1. Sampled, not 100% tested. T_A = 25°C.

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5.0V ±10%, GND = 0V

Symbol	Parameter	8256AH		Units
		Min	Max	
BUS PARAMETERS				
t_{LL}	ALE Pulse Width	50		ns
t_{CSL}	$\overline{CS}$ to ALE Setup Time	0		ns
t_{AL}	Address to ALE Setup Time	20		ns
t_{LA}	Address Hold Time after ALE	25		ns
t_{LC}	ALE to $\overline{RD}/\overline{WR}$	20		ns
t_{CC}	$\overline{RD}$, $\overline{WR}$, $\overline{INTA}$ Pulse Width	200		ns
t_{RD}	Data Valid from $\overline{RD}$(1)		120	ns

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5.0V ±10%, GND = 0V (Continued)

Symbol	Parameter	8256AH		Units
		Min	Max	
BUS PARAMETERS (Continued)				
t_{DF}	Data Float after $\overline{RD}$ (2)		50	ns
t_{DW}	Data Valid to $\overline{WR}$	150		ns
t_{WD}	Data Valid after $\overline{WR}$	50		ns
t_{CL}	$\overline{RD}/\overline{WR}$ Control to Latch Enable	25		ns
t_{LDR}	ALE to Data Valid		150	ns
t_{RST}	Reset Pulse Width	300		ns
t_{RV}	Recovery Time between $\overline{RD}/\overline{WR}$	500		ns
TIMER/COUNTER PARAMETERS				
t_{CPI}	Counter Input Cycle Time (P12, P13)	2.2		μs
t_{CPWH}	Counter Input Pulse Width High	1.1		μs
t_{CPWL}	Counter Input Pulse Width Low	1.1		μs
t_{TPI}	Counter Input ↑ to INT ↑ at Terminal Count		2.75	μs
t_{TIH}	LOAD Pulse High Time Counter 5	1.1		μs
t_{TIL}	LOAD Pulse Low Time Counter 5	1.1		μs
t_{PP}	Counter 5 Load before Next Clock Pulse on P13	1.1		μs
t_{CR}	External Count Clock ↑ to $\overline{RD}$ ↓ to Ensure Clock is Reflected in Count	2.2		μs
t_{RC}	$\overline{RD}$ ↑ to External Count Clock ↑ to Ensure Clock is not Reflected in Count	0		ns
t_{CW}	External Count Clock ↑ to $\overline{WR}$ ↑ to Ensure Count Written is Not Decremented	2.2		μs
t_{WC}	$\overline{WR}$ ↑ to External Count Clock to Ensure Count Written is Decremented	0		ns
INTERRUPT PARAMETERS				
t_{DEX}	EXTINT ↑ to INT ↑		200	ns
t_{DPI}	Interrupt Request on P17 ↑ to INT ↑		$2t_{CY}+500$	ns
t_{PI}	Pulse Width of Interrupt Request on P17	$t_{CY}+100$		ns
t_{HEA}	$\overline{INTA}$ ↑ or $\overline{RD}$ ↑ to EXTINT ↓	30		ns
t_{HIA}	$\overline{INTA}$ ↑ or $\overline{RD}$ ↑ to INT ↓		300	μs
SERIAL INTERFACE AND CLOCK PARAMETERS				
t_{CY}	Clock Period	195	1000	ns
t_{CLKH}	Clock High Pulse Width	65		ns
t_{CLKL}	Clock Low Pulse Width	65		ns
t_R	Clock Rise Time		20	ns
t_F	Clock Fall Time		20	ns

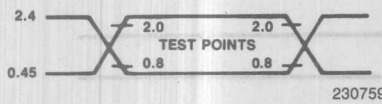
A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5.0V ±10%, GND = 0V (Continued)

Symbol	Parameter	8256AH		Units
		Min	Max	
SERIAL INTERFACE AND CLOCK PARAMETERS (Continued)				
t_{SCY}	Serial Clock Period (4)	975		ns
t_{SPD}	Serial Clock High (4)	350		ns
t_{SPW}	Serial Clock Low (4)	350		ns
t_{STD}	Internal Status Update Delay from Center of Stop Bit (5)		300	ns
t_{DTX}	$\overline{TxC}$ to TxD Data Valid		300	ns
t_{IRBF}	INT Delay from Center of First Stop Bit		$2t_{CY}+500$	ns
t_{ITBE}	INT Delay from Falling Edge of Transmit Clock at End of Start Bit		$2t_{CY}+500$	ns
t_{CTS}	Pulse Width for Single Character Transmission	(6)		
PARALLEL I/O PORT PARAMETERS				
t_{WP}	$\overline{WR}$ ↑ to P1/P2 Data Valid		0	ns
t_{PR}	P1/P2 Data Stable before $\overline{RD}$ ↓ (7)	300		ns
t_{RP}	P1/P2 Data Hold Time	50		ns
t_{AK}	$\overline{ACK}$ Pulse Width	150		ns
t_{ST}	Strobe Pulse Width	t_{SIB}		ns
t_{PS}	Data Setup to $\overline{STB}$ ↑	50		ns
t_{PH}	Data Hold after $\overline{STB}$ ↑	50		ns
t_{WOB}	$\overline{WR}$ ↑ to $\overline{OBF}$ ↑		250	ns
t_{AOB}	$\overline{ACK}$ ↓ to $\overline{OBF}$ ↓		250	ns
t_{SIB}	$\overline{STB}$ ↓ to $\overline{IBF}$ ↓		250	ns
t_{RI}	$\overline{RD}$ ↑ to $\overline{IBF}$ ↑		250	ns
t_{SIT}	$\overline{STB}$ ↑ to $\overline{INT}$ ↑		$2t_{CY}$ +500	ns
t_{AIT}	$\overline{ACK}$ ↑ to $\overline{INT}$ ↑		$2t_{CY}$ +500	ns
t_{AED}	$\overline{OBF}$ ↓ to $\overline{ACK}$ ↓ Delay	0		ns

NOTES:
1. C_L = pF all outputs.
2. Measured from logic "one" or "zero" to 1.5V at C_L = 150 pF.
3. P12, P13 are external clock inputs.
4. Note that RxC may be used as an input only in 1× mode, otherwise it will be an output.
5. The center of the Stop Bit will be the receiver sample time, as programmed by the modification register.
6. 1/16th bit length for 32×, 64×; 100 ns for 1×.
7. To ensure t_{RD} spec is met.

WAVEFORMS

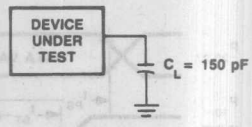
A.C. TESTING INPUT, OUTPUT WAVEFORM



230759-5

A.C. testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

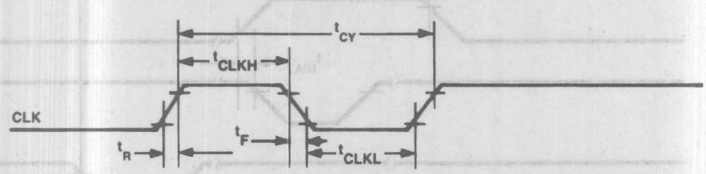
A.C. TESTING LOAD CIRCUIT



230759-6

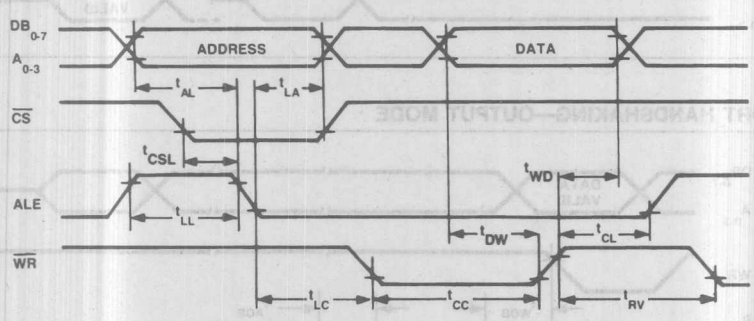
C_L = 150 pF
C_L Includes Jig Capacitance

SYSTEM CLOCK



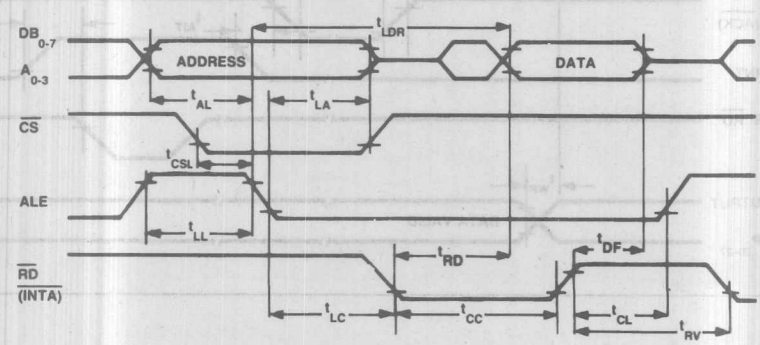
230759-7

WRITE CYCLE



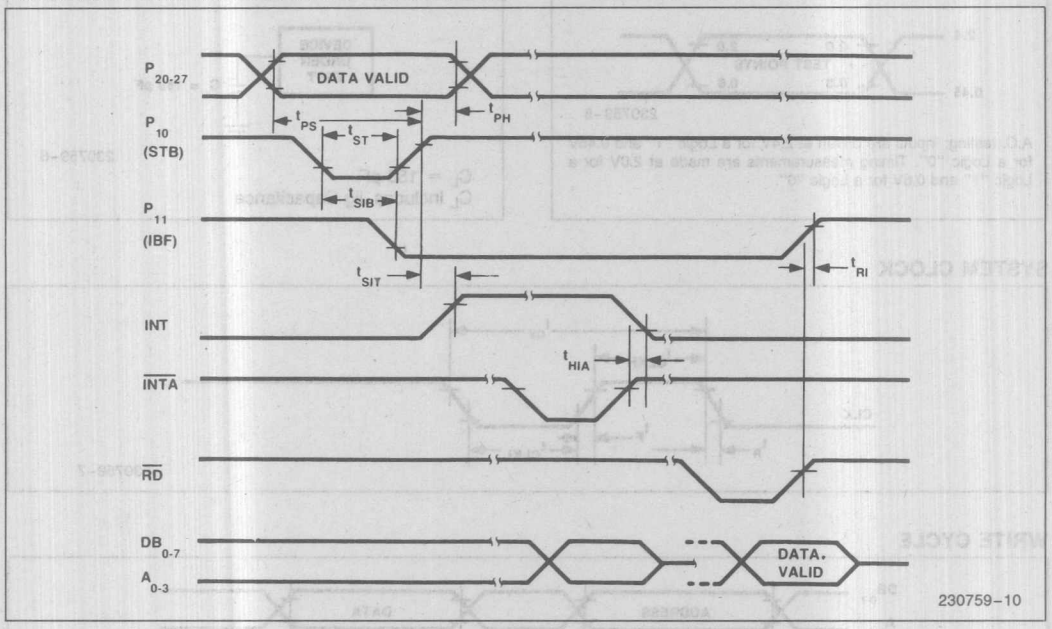
230759-8

READ CYCLE

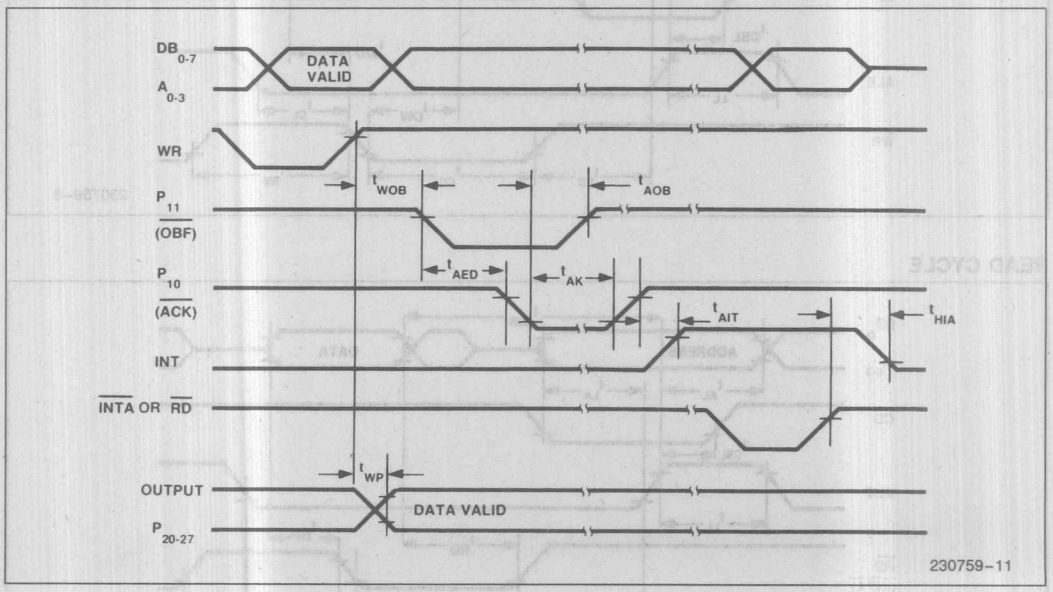


230759-9

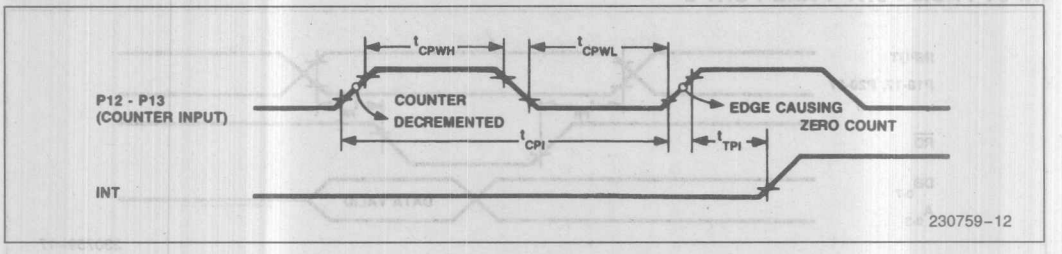
PARALLEL PORT HANDSHAKING—INPUT MODE



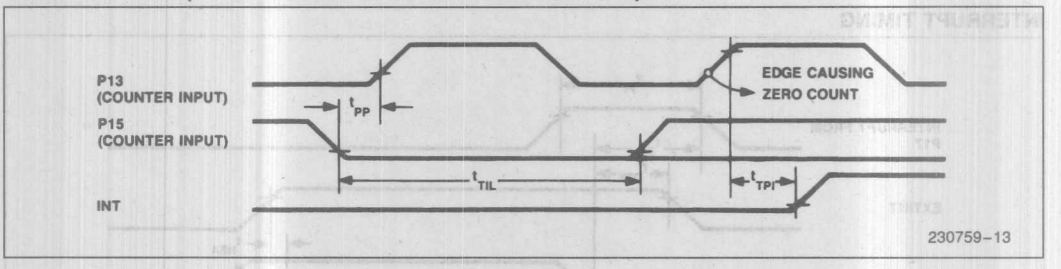
PARALLEL PORT HANDSHAKING—OUTPUT MODE



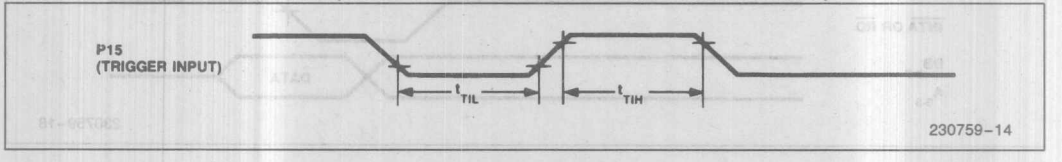
COUNT PULSE TIMINGS



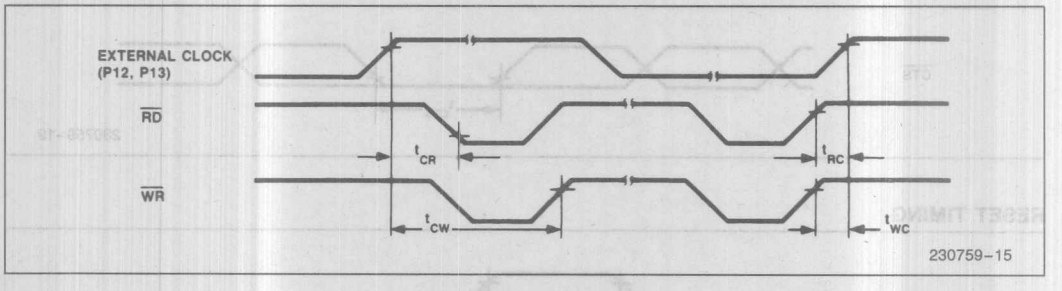
LOADING TIMER (OR CASCADED COUNTER/TIMER 3 AND 5)



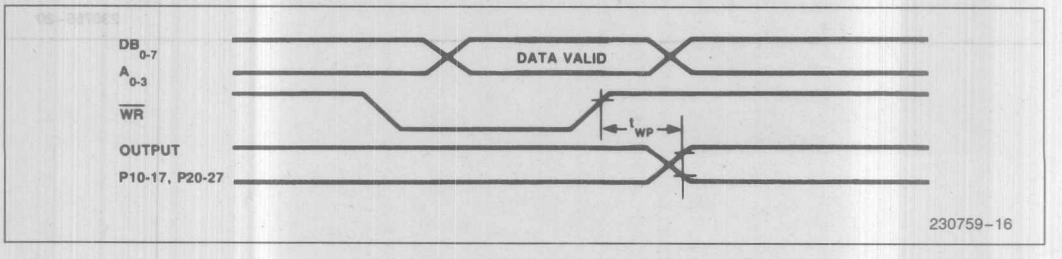
TRIGGER PULSE FOR TIMER 5 (CASCADED EVENT COUNTER/TIMER 3 AND 5)



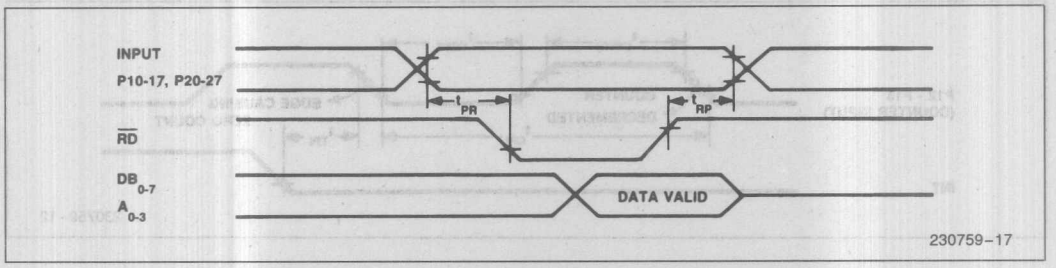
COUNTER TIMER TIMING



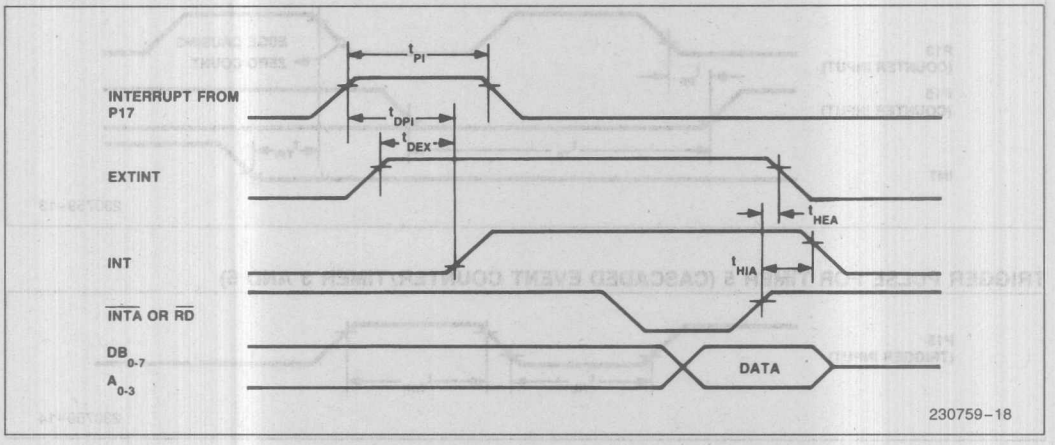
OUTPUT FROM PORT 1 AND PORT 2



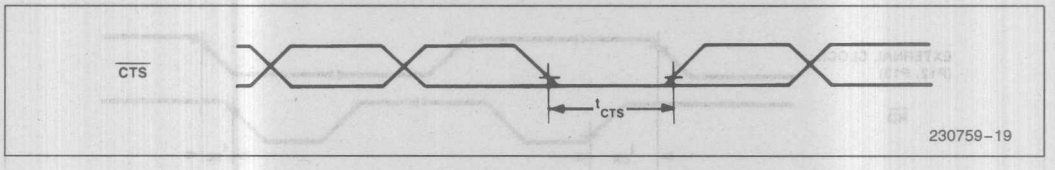
INPUT FROM PORT 1 AND PORT 2



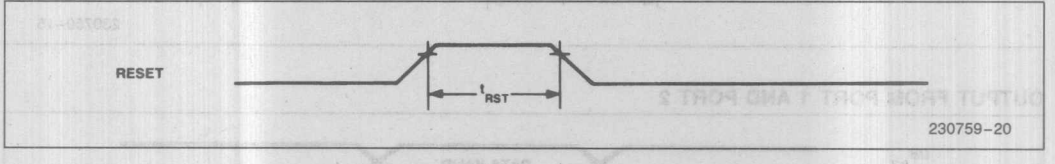
INTERRUPT TIMING



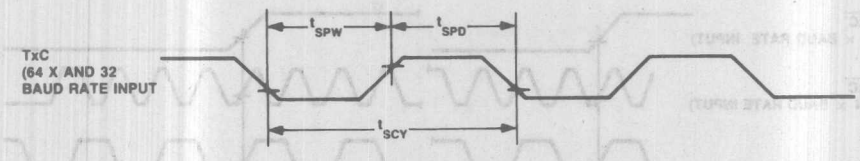
CTS FOR SINGLE CHARACTER TRANSMISSION



RESET TIMING

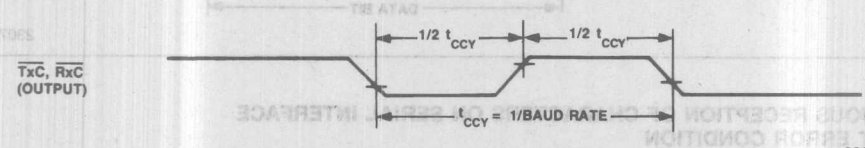


EXTERNAL BAUD RATE CLOCK FOR SERIAL INTERFACE



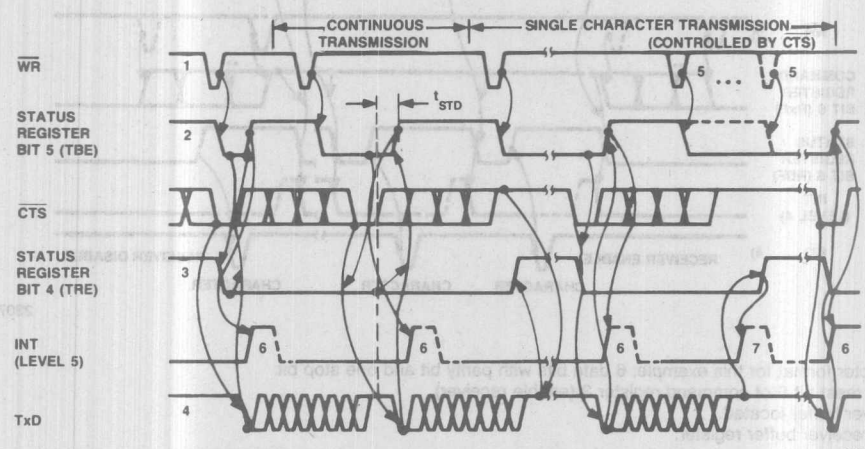
230759-21

TRANSMITTER AND RECEIVER CLOCK FROM INTERNAL CLOCK SOURCE



230759-22

TRANSMISSION OF CHARACTERS ON SERIAL INTERFACE

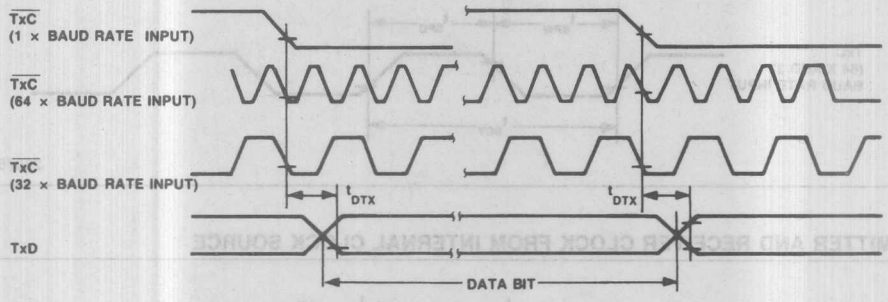


230759-23

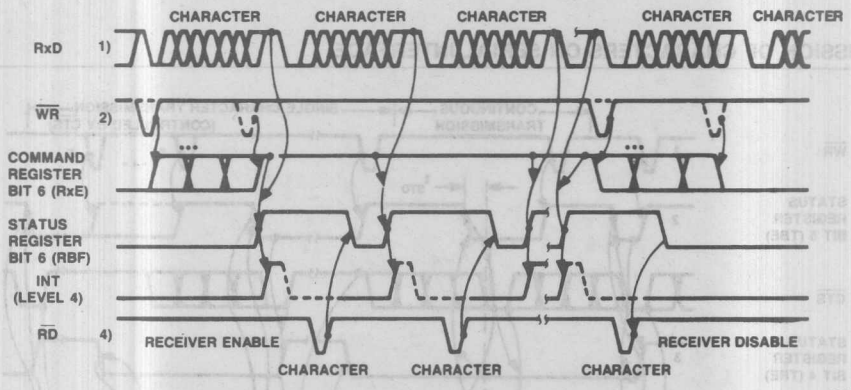
NOTES:
1. Load transmitter buffer register.
2. Transmitter buffer register is empty.
3. Transmitter register is empty.
4. Character format for this example: 7 Data Bits with Parity Bit and 2 Stop Bits.
5. Loading of transmitter buffer register must be complete before $\overline{CTS}$ goes low.
6. Interrupt due to transmitter buffer register empty.
7. Interrupt due to transmitter register empty.

No status bits are altered when $\overline{RD}$ is active.

DATA BIT OUTPUT ON SERIAL INTERFACE

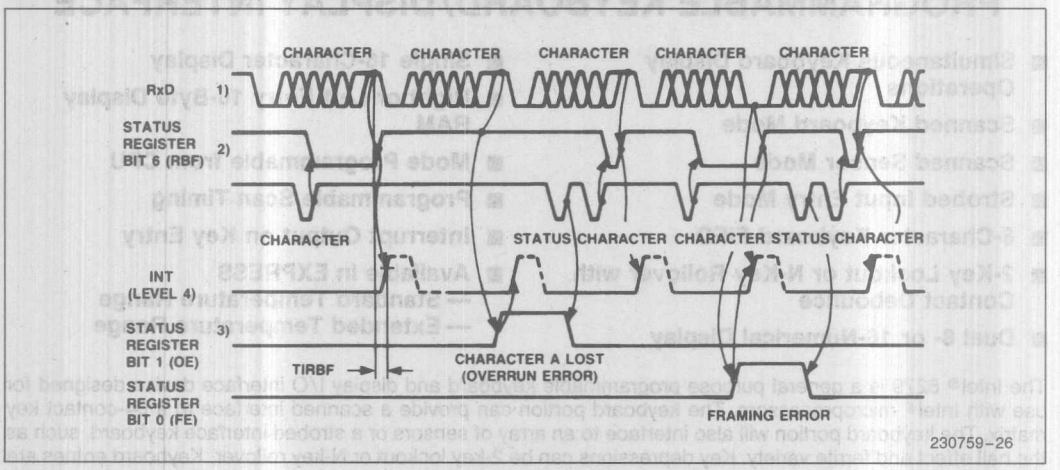


CONTINUOUS RECEPTION OF CHARACTERS ON SERIAL INTERFACE WITHOUT ERROR CONDITION



NOTES:
1. Character format for this example: 6 data bits with parity bit and one stop bit.
2. Set or reset bit 6 of command register 3 (enable receiver).
3. Receiver buffer located.
4. Read receiver buffer register.

ERROR CONDITIONS DURING RECEPTION OF CHARACTERS ON THE SERIAL INTERFACE



NOTES:
1. Character format for this example: 6 data bits without parity and one stop bit.
2. Receiver buffer register loaded.
3. Overrun error.
4. Framing error.
5. Interrupt from receiver buffer register loading.
6. Interrupt from overrun error.
7. Interrupt from framing error and loading receiver buffer register.

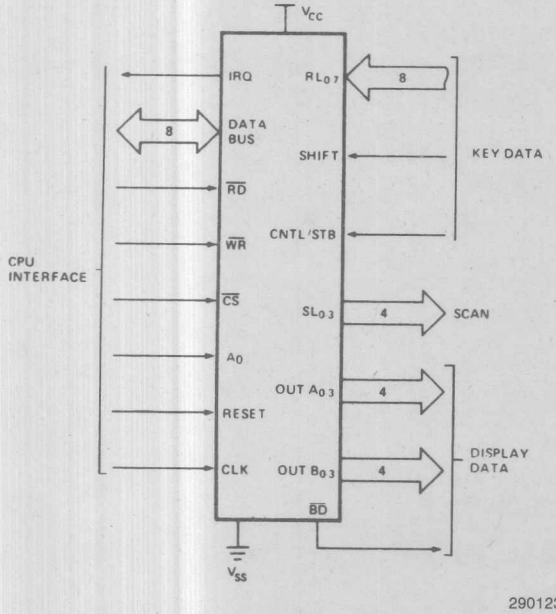
No status bits are altered when $\overline{RD}$ is active.

8279/8279-5
PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- Simultaneous Keyboard Display Operations
- Scanned Keyboard Mode
- Scanned Sensor Mode
- Strobed Input Entry Mode
- 8-Character Keyboard FIFO
- 2-Key Lockout or N-Key Rollover with Contact Debounce
- Dual 8- or 16-Numerical Display
- Single 16-Character Display
- Right or Left Entry 16-Byte Display RAM
- Mode Programmable from CPU
- Programmable Scan Timing
- Interrupt Output on Key Entry
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

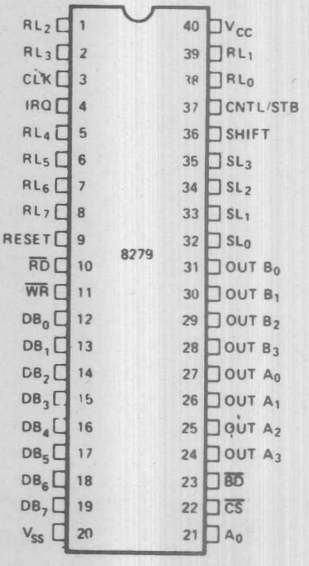
The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as the hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16x8 display RAM which can be organized into dual 16x4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address.



290123-1

Figure 1. Logic Symbol



290123-2

Figure 2. Pin Configuration

September 1987
Order Number: 290123-002

HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

Table 1. Pin Description

Symbol	Pin No.	Name and Function
DB_0–DB_7	19–12	**BI-DIRECTIONAL DATA BUS:** All data and commands between the CPU and the 8279 are transmitted on these lines.
CLK	3	**CLOCK:** Clock from system used to generate internal timing.
RESET	9	**RESET:** A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display—left entry. 2) Encoded scan keyboard—2 key lockout. Along with this the program clock prescaler is set to 31.
CS	22	**CHIP SELECT:** A low on this pin enables the interface functions to receive or transmit.
A_0	21	**BUFFER ADDRESS:** A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.
$\overline{RD}$, $\overline{WR}$	10–11	**INPUT/OUTPUT READ AND WRITE:** These signals enable the data buffers to either send data to the external bus or receive it from the external bus.
IRQ	4	**INTERRUPT REQUEST:** In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.
V_{SS}, V_{CC}	20, 40	**GROUND AND POWER SUPPLY PINS.**
SL_0–SL_3	32–35	**SCAN LINES:** Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).
RL_0–RL_7	38, 39, 1, 2, 5–8	**RETURN LINE:** Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed Input mode.
SHIFT	36	**SHIFT:** The shift input status is stored along with the key position on key closure in the Scanned Keyboard modes. It has an active internal pullup to keep it high until a switch closure pulls it low.
CNTL/STB	37	**CONTROL/STROBED INPUT MODE:** For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed Input mode. (Rising Edge). It has an active internal pullup to keep it high until a switch closure pulls it low.
OUT A_0–OUT A_3 OUT B_0–OUT B_3	27-24 31–28	**OUTPUTS:** These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL_0–SL_3) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8-bit port.
$\overline{BD}$	23	**BLANK DISPLAY:** This output is used to blank the display during digit switching or by a display blanking command.

FUNCTIONAL DESCRIPTION

Since data input and display are an integral part of many microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors.

The 8279 has two sections: keyboard and display. The keyboard section can interface to regular typewriter style keyboards or random toggle or thumb switches. The display section drives alphanumeric displays or a bank of indicator lights. Thus the CPU is relieved from scanning the keyboard or refreshing the display.

The 8279 is designed to directly connect to the microprocessor bus. The CPU can program all operating modes for the 8279. These modes include:

Input Modes

- Scanned Keyboard—with encoded (8 x 8 key keyboard) or decoded (4 x 8 key keyboard) scan lines. A key depression generates a 6-bit encoding of key position. Position and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or N-key rollover.
- Scanned Sensor Matrix—with encoded (8 x 8 matrix switches) or decoded (4 x 8 matrix switches) scan lines. Key status (open or closed) stored in RAM addressable by CPU.
- Strobed Input—Data on return lines during control line strobe is transferred to FIFO.

Output Modes

- 8 or 16 character multiplexed displays that can be organized as dual 4-bit or single 8-bit ($B_0 = D_0$, $A_3 = D_7$).
- Right entry or left entry display formats.

Other features of the 8279 include:

- Mode programming from the CPU.
- Clock Prescaler
- Interrupt output to signal CPU when there is keyboard or sensor data available.
- An 8 byte FIFO to store keyboard information.
- 16 byte internal Display RAM for display refresh. This RAM can also be read by the CPU.

PRINCIPLES OF OPERATION

The following is a description of the major elements of the 8279 Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 3.

I/0 Control and Data Buffers

The I/O control section uses the $\overline{CS}$, A_0, $\overline{RD}$ and $\overline{WR}$ lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by $\overline{CS}$. The character of the information, given or desired by the CPU, is identified by A_0. A logic one means the information is a command or status. A logic zero means the information is data. $\overline{RD}$ and $\overline{WR}$ determine the direction of data flow through the Data Buffers. The Data Buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected ($\overline{CS} = 1$), the devices are in a high impedance state. The drivers input during $\overline{WR} \bullet \overline{CS}$ and output during $\overline{RD} \bullet \overline{CS}$.

Control and Timing Registers and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. The modes are programmed by presenting the proper command on the data lines with $A_0 = 1$ and then sending a $\overline{WR}$. The command is latched on the rising edge of $\overline{WR}$. The command is then decoded and the appropriate function is set. The timing control contains the basic timing counter chain. The first counter is a ÷ N prescaler that can be programmed to yield an internal frequency of 100 kHz which gives a 5.1 ms keyboard scan time and a 10.3 ms debounce time. The other counters divide down the basic internal frequency to provide the proper key scan, row scan, keyboard matrix scan, and display scan times.

Scan Counter

The scan counter has two modes. In the encoded mode, the counter provides a binary count that must be externally decoded to provide the scan lines for the keyboard and display. In the decoded mode, the scan counter decodes the least significant 2 bits and provides a decoded 1 of 4 scan. Note that when the keyboard is in decoded scan, so is the display. This means that only the first 4 characters in the Display RAM are displayed.

In the encoded mode, the scan lines are active high outputs. In the decoded mode, the scan lines are active low outputs.

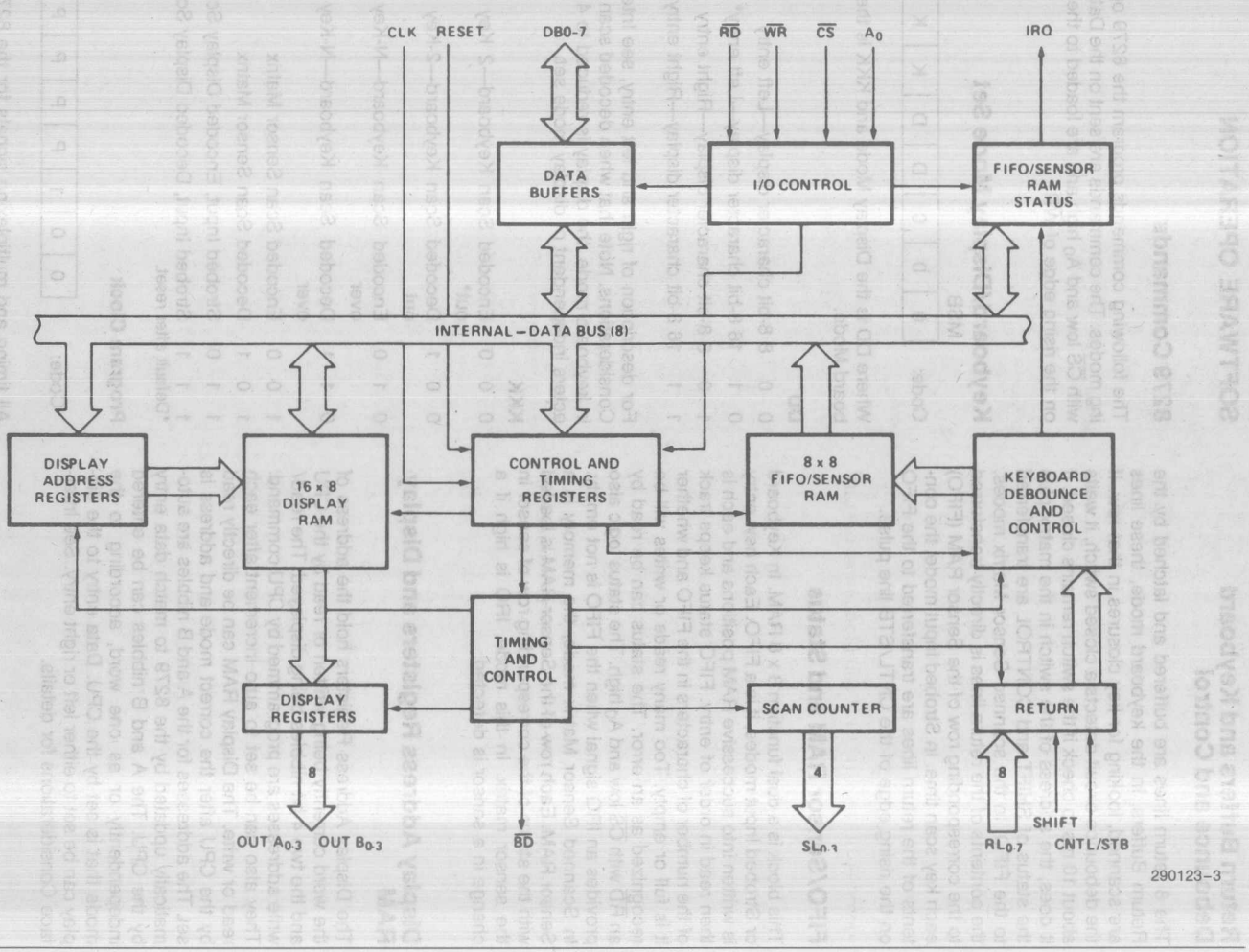


Figure 3. Internal Block Diagram

Return Buffers and Keyboard Debounce and Control

The 8 return lines are buffered and latched by the Return Buffers. In the keyboard mode, these lines are scanned, looking for key closures in that row. If the debounce circuit detects a closed switch, it waits about 10 ms to check if the switch remains closed. If it does, the address of the switch in the matrix plus the status of SHIFT and CONTROL are transferred to the FIFO. In the scanned Sensor Matrix modes, the contents of the return lines is directly transferred to the corresponding row of the Sensor RAM (FIFO) each key scan time. In Strobed Input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CNTL/STB line pulse.

FIFO/Sensor RAM and Status

This block is a dual function 8 x 8 RAM. In Keyboard or Strobed Input modes, it is a FIFO. Each new entry is written into successive RAM positions and each is then read in order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognized as an error. The status can be read by an $\overline{RD}$ with $\overline{CS}$ low and A_0 high. The status logic also provides an IRQ signal when the FIFO is not empty. In Scanned Sensor Matrix mode, the memory is a Sensor RAM. Each row of the Sensor RAM is loaded with the status of the corresponding row of sensor in the sensor matrix. In this mode, IRQ is high if a change in a sensor is detected.

Display Address Registers and Display RAM

The Display Address Registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed. The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The Display RAM can be directly read by the CPU after the correct mode and address is set. The addresses for the A and B nibbles are automatically updated by the 8279 to match data entry by the CPU. The A and B nibbles can be entered independently or as one word, according to the mode that is set by the CPU. Data entry to the display can be set to either left or right entry. See Interface Considerations for details.

SOFTWARE OPERATION

8279 Commands

The following commands program the 8279 operating modes. The commands are sent on the Data Bus with $\overline{CS}$ low and A_0 high and are loaded to the 8279 on the rising edge of $\overline{WR}$.

Keyboard/Display Mode Set

	MSB							LSB
Code:	0	0	0	D	D	K	K	K

Where DD is the Display Mode and KKK is the Keyboard Mode.

DD

0	0	8 8-bit character display—Left entry
0	1	16 8-bit character display—Left entry*
1	0	8 8-bit character display—Right entry
1	1	16 8-bit character display—Right entry

For description of right and left entry, see Interface Considerations. Note that when decoded scan is set in keyboard mode, the display is reduced to 4 characters independent of display mode set.

KKK

0	0	0	Encoded Scan Keyboard—2 Key Lockout*
0	0	1	Decoded Scan Keyboard—2-Key Lockout
0	1	0	Encoded Scan Keyboard—N-Key Rollover
0	1	1	Decoded Scan Keyboard—N-Key Rollover
1	0	0	Encoded Scan Sensor Matrix
1	0	1	Decoded Scan Sensor Matrix
1	1	0	Strobed Input, Encoded Display Scan
1	1	1	Strobed Input, Decoded Display Scan

*Default after reset.

Program Clock

Code:	0	0	1	P	P	P	P	P

All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock (pin 3) by a programmable integer. Bits PPPPP determine the value of this integer which ranges from 2 to 31. Choosing a divisor that yields 100 kHz will give the specified scan and

debounce times. For instance, if Pin 3 of the 8279 is being clocked by a 2 MHz signal, PPPPP should be set to 10100 to divide the clock by 20 to yield the proper 100 kHz operating frequency.

Read FIFO/Sensor RAM

Code:	0	1	0	AI	X	A	A	A

X = Don't Care

The CPU sets the 8279 for a read of the FIFO/Sensor RAM by first writing this command. In the Scan Keyboard Mode, the Auto-Increment flag (AI) and the RAM address bits (AAA) are irrelevant. The 8279 will automatically drive the data bus for each subsequent read (A_0 = 0) in the same sequence in which the data first entered the FIFO. All subsequent reads will be from the FIFO until another command is issued.

In the Sensor Matrix Mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM. If the AI flag is set (AI = 1), each successive read will be from the subsequent row of the sensor RAM.

Read Display RAM

Code:	0	1	1	AI	A	A	A	A

The CPU sets up the 8279 for a read of the Display RAM by first writing this command. The address bits AAAA select one of the 16 rows of the Display RAM. If the AI flag is set (A1 = 1), this row address will be incremented after each following read *or write* to the Display RAM. Since the same counter is used for both reading and writing, this command sets the next read or *write* address and the sense of the Auto-Increment mode for both operations.

Write Display RAM

Code:	1	0	0	AI	A	A	A	A

The CPU sets up the 8279 for a write to the Display RAM by first writing this command. After writing the command with A_0 = 1, all subsequent writes with A_0 = 0 will be to the Display RAM. The addressing and Auto-Increment functions are identical to those for the Read Display RAM. However, this command does not affect the source of subsequent Data Reads; the CPU will read from whichever RAM (Display of FIFO/Sensor) which was last specified. If, indeed, the Display RAM was last specified, the Write Display RAM will, nevertheless, change the next Read location.

Display Write Inhibit/Blanking

					A	B	A	B
Code:	1	0	1	X	IW	IW	BL	BL

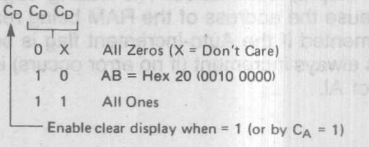
The IW Bits can be used to mask nibble A and nibble B in applications requiring separate 4-bit display ports. By setting the IW flag (IW = 1) for one of the ports, the port becomes marked so that entries to the Display RAM from the CPU do not affect that port. Thus, if each nibble is input to a BCD decoder, the CPU may write a digit to the Display RAM without affecting the other digit being displayed. It is important to note that bit B_0 corresponds to bit D_0 on the CPU bus, and that bit A_3 corresponds to bit D_7.

If the user wishes to blank the display, the BL flags are available for each nibble. The last Clear command issued determines the code to be used as a "blank." This code defaults to all zeros after a reset. Note that both BL flags must be set to blank a display formatted with a single 8-bit port.

Clear

Code:	1	1	0	C_D	C_D	C_D	C_F	C_A

The C_D bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as follows:



290123-13

During the time the Display RAM is being cleared (~160 μs), it may not be written to. The most significant bit of the FIFO status word is set during this time. When the Display RAM becomes available again, it automatically resets.

If the C_F bit is asserted (C_F = 1), the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

C_A, the Clear All bit, has the combined effect of C_D and C_F; it uses the C_D clearing code on the Display RAM and also clears FIFO status. Furthermore, it resynchronizes the internal timing chain.

End Interrupt/Error Mode Set

Code:	1	1	1	E	X	X	X	X

X = Don't care

For the sensor matrix modes this command lowers the IRQ line and enables further writing into RAM. (The IRQ line would have been raised upon the detection of a change in a sensor value. This would have also inhibited further writing into the RAM until reset).

For the N-key rollover mode—if the E bit is programmed to "1" the chip will operate in the special Error mode. (For further details, see Interface Considerations Section.)

Status Word

The status word contains the FIFO status, error, and display unavailable signals. This word is read by the CPU when A_0 is high and $\overline{CS}$ and $\overline{RD}$ are low. See Interface Considerations for more detail on status word.

Data Read

Data is read when A_0, $\overline{CS}$ and $\overline{RD}$ are all low. The source of the data is specified by the Read FIFO or Read Display commands. The trailing edge of $\overline{RD}$ will cause the address of the RAM being read to be incremented if the Auto-Increment flag is set. FIFO reads always increment (if no error occurs) independent of AI.

Data Write

Data that is written with A_0, $\overline{CS}$ and $\overline{WR}$ low is always written to the Display RAM. The address is specified by the latest Read Display or Write Display command. Auto-Incrementing on the rising edge of $\overline{WR}$ occurs if AI is set by the latest display command.

INTERFACE CONSIDERATIONS

Scanned Keyboard Mode, 2-Key Lockout

There are three possible combinations of conditions that can occur during debounce scanning. When a key is depressed, the debounce logic is set. Other depressed keys are looked for during the next two scans. If none are encountered, it is a single key depression and the key position is entered into the FIFO along with the status of CNTL and SHIFT lines. If the FIFO was empty, IRQ will be set to signal the CPU that there is an entry in the FIFO. If the FIFO was full, the key will not be entered and the error flag will be set. If another closed switch is encountered, no entry to the FIFO can occur. If all other keys are released before ths one, then it will be entered to the FIFO. If this key is released before any other, it will be entirely ignored. A key is entered to the FIFO only once per depression, no matter how many keys were pressed along with it or in what order they were released. If two keys are depressed within the debounce cycle, it is a simultaneous depression. Neither key will be recognized until one key remains depressed alone. The last key will be treated as a single key depression.

Scanned Keyboard Mode, N-Key Rollover

With N-key Rollover each key depression is treated independently from all others. When a key is depressed, the debounce circuit waits 2 keyboard scans and then checks to see if the key is still down. If it is , the key is entered into the FIFO. Any number of keys can be depressed and another can be recognized and entered into the FIFO. If a simultaneous depression occurs, the keys are recognized and entered according to the order the keyboard scan found them.

Scanned Keyboard—Special Error Modes

For N-key rollover mode the user can program a special error mode. This is done by the "End Interrupt/Error Mode Set" command. The debounce cycle and key-validity check are as in normal N-key mode. If during a *single debounce cycle*, two keys are found depressed, this is considered a simultaneous multiple depression, and sets an error flag. This flag will prevent any further writing into the FIFO and will set interrupt (if not yet set). The error flag could be read in this mode by reading the FIFO STATUS word. (See "FIFO STATUS" for further details.) The error flag is reset by sending the normal CLEAR command with CF = 1.

Sensor Matrix Mode

In Sensor Matrix mode, the debounce logic is inhibited. The status of the sensor switch is inputted directly to the Sensor RAM. In this way the Sensor RAM keeps an image of the state of the switches in the sensor matrix. Although debouncing is not provided, this mode has the advantage that the CPU knows how long the sensor was closed and when it

was released. A keyboard mode can only indicate a validated closure. To make the software easier, the designer should functionally group the sensors by row since this is the format in which the CPU will read them.

The IRQ line goes high if any sensor value change is detected at the end of a sensor matrix scan. The IRQ line is cleared by the first data read operation if the Auto-Increment flag is set to zero, or by the End Interrupt command if the Auto-Increment flag is set to one.

NOTE:
Multiple changes in the matrix Addressed by ($SL_{0-3} = 0$) may cause multiple interrupts. ($SL_0 = 0$ in the Decoded Mode.) Reset may cause the 8279 to see multiple changes.

Data Format

In the Scanned Keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines (non-inverted). CNTL is the MSB of the character and SHIFT is the next most significant bit. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate to which return line the key was connected.

MSB							LSB
CNTL	SHIFT	SCAN			RETURN		

SCANNED KEYBOARD DATA FORMAT

In Sensor Matrix mode, the data on the return lines is entered directly in the row of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each switch position maps directly to a Sensor RAM position. The SHIFT and CNTL inputs are ignored in this mode. Note that switches are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the scan lines can enter data to the return line inputs. Eight multiplexed input ports could be tied to the return lines and scanned by the 8279.

MSB							LSB
RL_7	RL_6	RL_5	RL_4	RL_3	RL_2	RL_1	RL_0

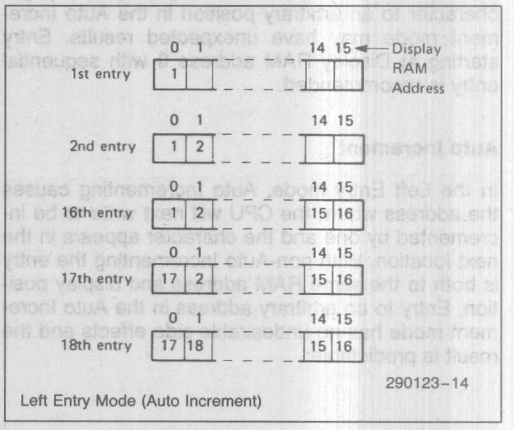
In Strobed Input mode, the data is also entered to the FIFO from the return lines. The data is entered by the rising edge of a CNTL/STB line pulse. Data can come from another encoded keyboard or simple switch matrix. The return lines can also be used as a general purpose strobed input.

MSB							LSB
RL_7	RL_6	RL_5	RL_4	RL_3	RL_2	RL_1	RL_0

Display

Left Entry

Left Entry mode is the simplest display format in that each display position directly corresponds to a byte (or nibble) in the Display RAM. Address 0 in the RAM is the left-most display character and address 15 (or address 7 in 8 character display) is the right most display character. Entering characters from position zero causes the display to fill from the left. The 17th (9th) character is entered back in the left most position and filling again proceeds from there.

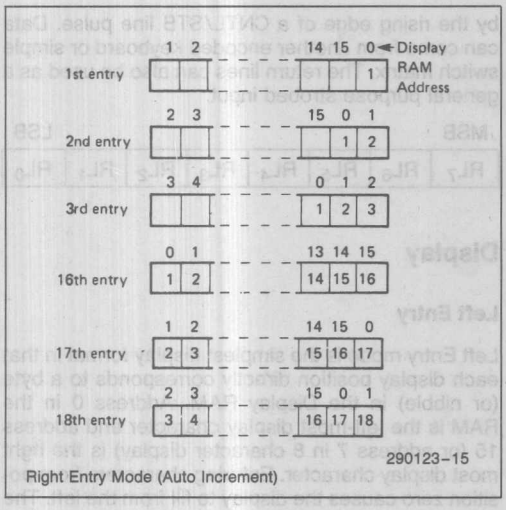


290123-14

Left Entry Mode (Auto Increment)

Right Entry

Right entry is the method used by most electronic calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost.

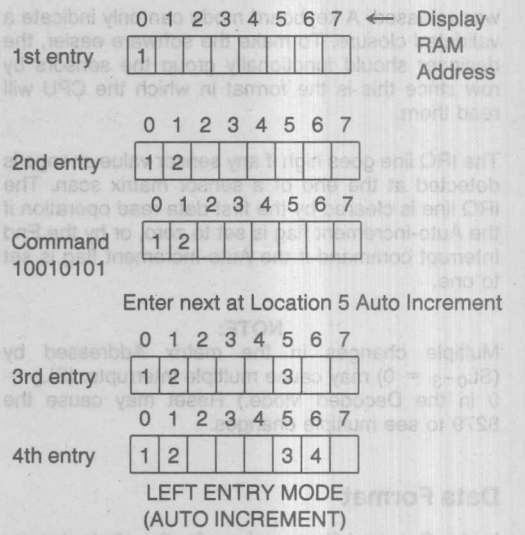


Right Entry Mode (Auto Increment)

Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM address 0 with sequential entry is recommended.

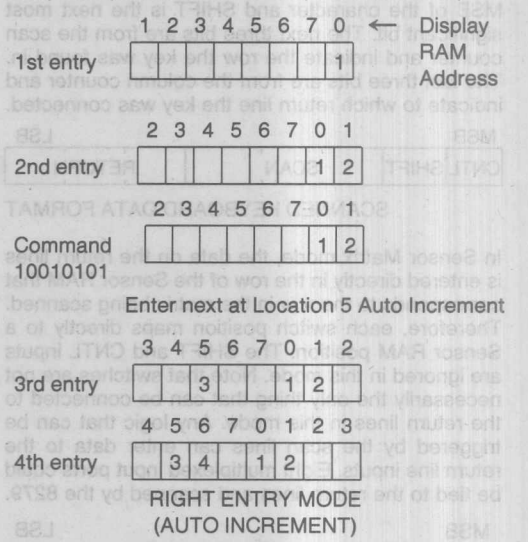
Auto Increment

In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Auto Increment mode has no undesirable side effects and the result is predictable:



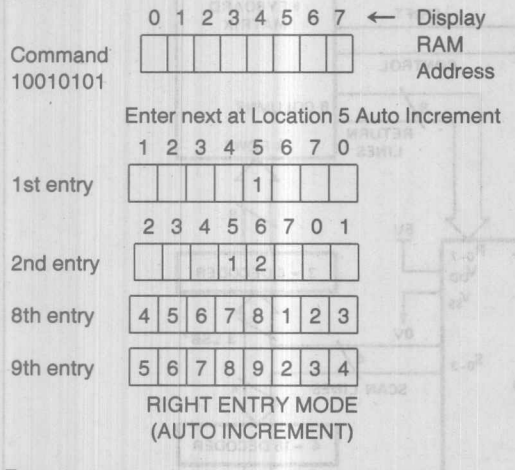
LEFT ENTRY MODE
(AUTO INCREMENT)

In the Right Entry mode, Auto Incrementing and non Incrementing have the same effect as in the Left Entry except if the address sequence is interrupted.



RIGHT ENTRY MODE
(AUTO INCREMENT)

Starting at an arbitrary location operates as shown below:



Entry appears to be from the intial entry point.

8/16 Character Display Formats

If the display mode is set to an 8 character display, the on duty-cycle is double what it would be for a 16 character display (e.g., 5.1 ms scan time for 8 characters vs. 10.3 ms for 16 characters with 100 kHz internal frequency).

G. FIFO Status

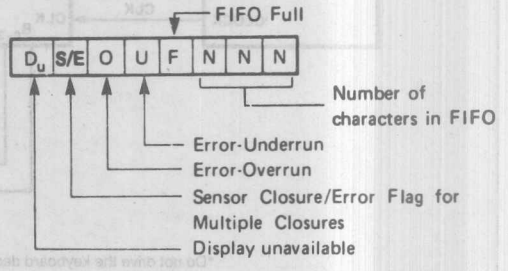
FIFO status is used in the Keyboard and Strobed Input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. There are two types of errors possible: overrun and underrun. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

The FIFO status word also has a bit to indicate that the Display RAM was unavailable because a Clear Display or Clear All command had not completed its clearing operation.

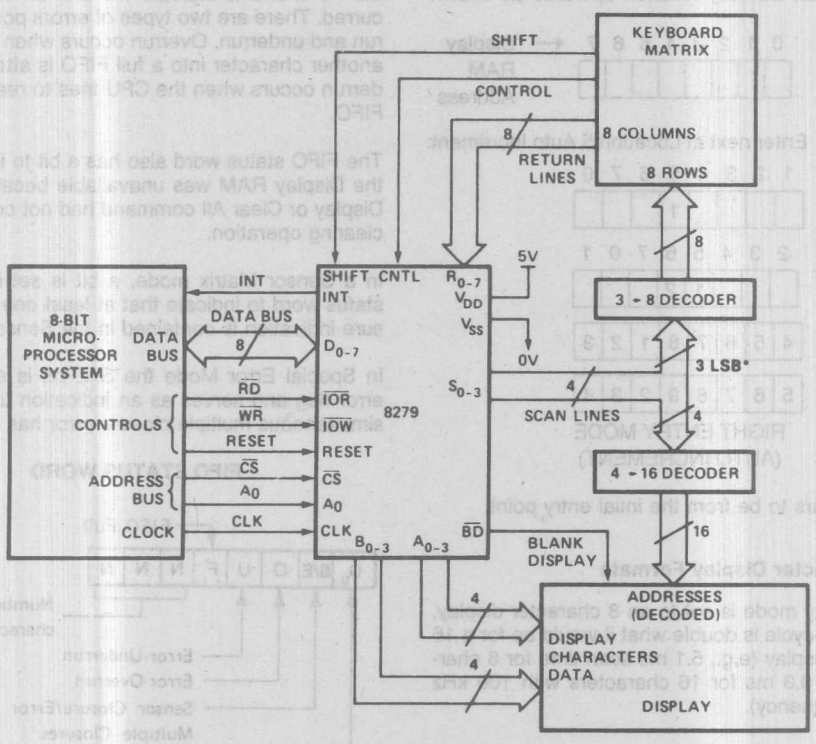
In a Sensor Matrix mode, a bit is set in the FIFO status word to indicate that at least one sensor closure indication is contained in the Sensor RAM.

In Special Error Mode the S/E bit is showing the error flag and serves as an indication to whether a simultaneous multiple closure error has occurred.

FIFO STATUS WORD



290123-4



*Do not drive the keyboard decoder with the MSB of the scan lines.

Figure 4. System Block Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature ..... 0°C to 70°C
Storage Temperature ..... −65°C to 125°C
Voltage on any Pin with Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C V_{SS} = 0V (Note 3)*

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL1}	Input Low Voltage for Return Lines	−0.5	1.4	V	
V_{IL2}	Input Low Voltage for All Others	−0.5	0.8	V	
V_{IH1}	Input High Voltage for Return Lines	2.2		V	
V_{IH2}	Input High Voltage for All Others	2.0		V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH1}	Output High Voltage on Interrupt Line	3.5		V	(Note 2)
V_{OH2}	Other Outputs	2.4			I_{OH} = −400 μA 8279-5 −100 μA 8279
I_{IL1}	Input Current on Shift, Control and Return Lines		+10 −100	μA μA	$V_{IN} = V_{CC}$ V_{IN} = 0V
I_{IL2}	Input Leakage Current on All Others		±10	μA	$V_{IN} = V_{CC}$ to 0V
I_{OFL}	Output Float Leakage		±10	μA	$V_{OUT} = V_{CC}$ to 0.45V
I_{CC}	Power Supply Current		120	mA	
C_{IN}	Input Capacitance		10	pF	f_C = 1 MHz Unmeasured Pins Returned to V_{SS}(6)
C_{OUT}	Output Capacitance		20	pF	

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{SS} = 0V (Note 3)*

Bus Parameters

READ CYCLE

Symbol	Parameter	8279		8279-5		Unit
		Min	Max	Min	Max	
t_{AR}	Address Stable Before $\overline{READ}$	50		0		ns
t_{RA}	Address Hold Time for $\overline{READ}$	5		0		ns
t_{RR}	$\overline{READ}$ Pulse Width	420		250		ns
t_{RD}(4)	Data Delay from $\overline{READ}$		300		150	ns
t_{AD}(4)	Address to Data Valid		450		250	ns
t_{DF}	$\overline{READ}$ to Data Floating	10	100	10	100	ns
t_{RCY}	Read Cycle Time	1		1		μs
t_{AW}	Address Stable Before $\overline{WRITE}$	50		0		ns
t_{WA}	Address Hold Time for $\overline{WRITE}$	20		0		ns

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8279		8279-5		Unit
		Min	Max	Min	Max	
t_{WW}	$\overline{WRITE}$ Pulse Width	400		250		ns
t_{DW}	Data Set Up Time for $\overline{WRITE}$	300		150		ns
t_{WD}	Data Hold Time for $\overline{WRITE}$	40		0		ns
t_{WCY}	Write Cycle Time	1		1		μs

OTHER TIMINGS

Symbol	Parameter	8279		8279-5		Unit
		Min	Max	Min	Max	
$t_{\phi W}$	Clock Pulse Width	230		120		ns
t_{CY}	Clock Period	500		320		ns

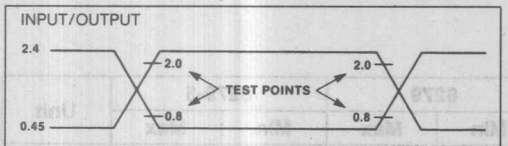
Keyboard Scan Time.....5.1 ms
Keyboard Debounce Time.....10.3 ms
Key Scan Time.....80 μs
Display Scan Time.....10.3 ms

Digit-on Time.....480 μs
Blanking Time.....160 μs
Internal Clock Cycle(5).....10 μs

NOTES:
1. 8279, I_{OL} = 1.6 mA; 8279-5, I_{OL} = 2.2 mA.
2. I_{OH} = −100 μA
3. 8279, V_{CC} = +5V ±5%; 8279-5, V_{CC} = +5V ±10%
4. 8279, C_L = 100 pF; 8279-5, C_L = 150 pF.
5. The Prescaler should be programmed to provide a 10 μs internal clock cycle.
6. Sampled not 100% tested. T_A = 25°C.

* For Extended Temperature EXPRESS, use M8279A electrical parameters.

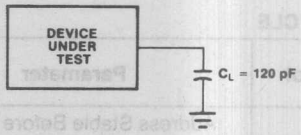
A.C. TESTING INPUT, OUTPUT WAVEFORM



290123-6

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT

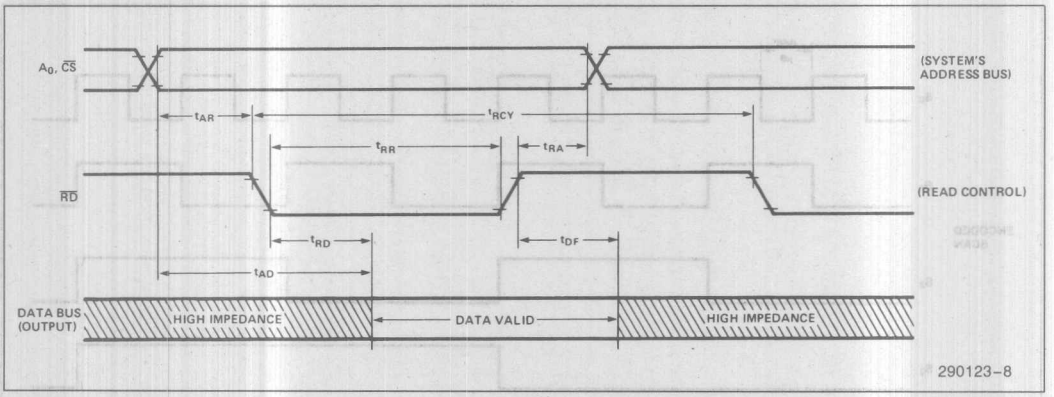


290123-7

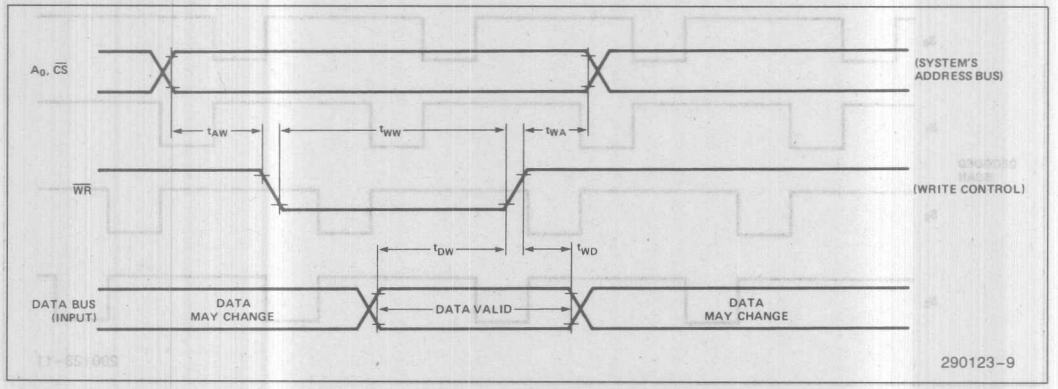
C_L = 120 pF
C_L Includes Jig Capacitance

WAVEFORMS

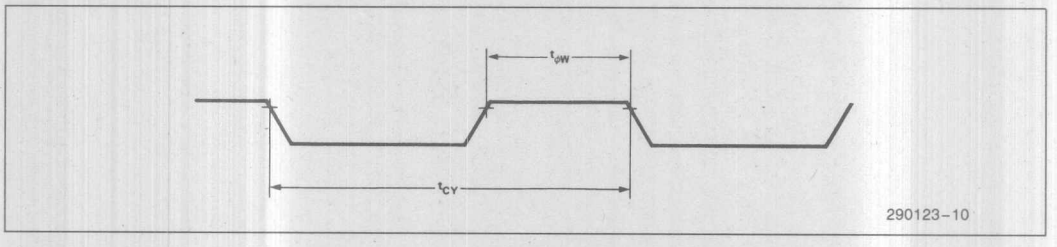
READ OPERATION



WRITE OPERATION

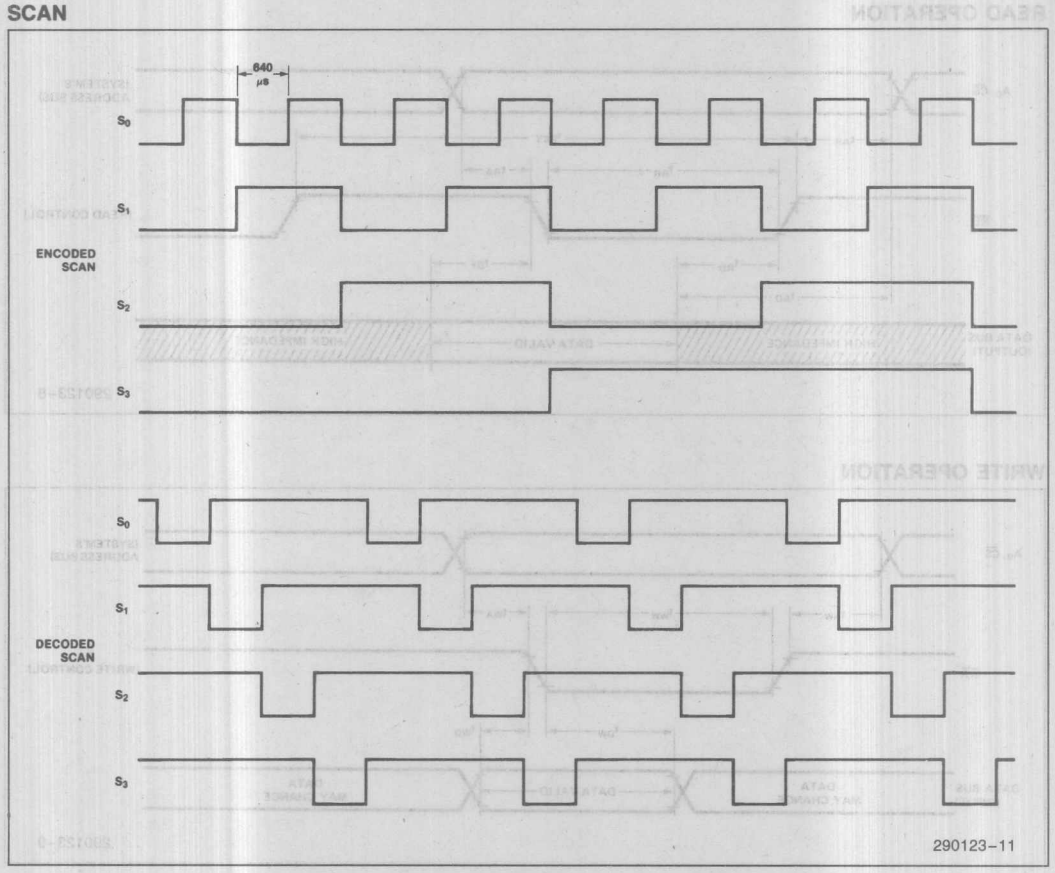


CLOCK INPUT



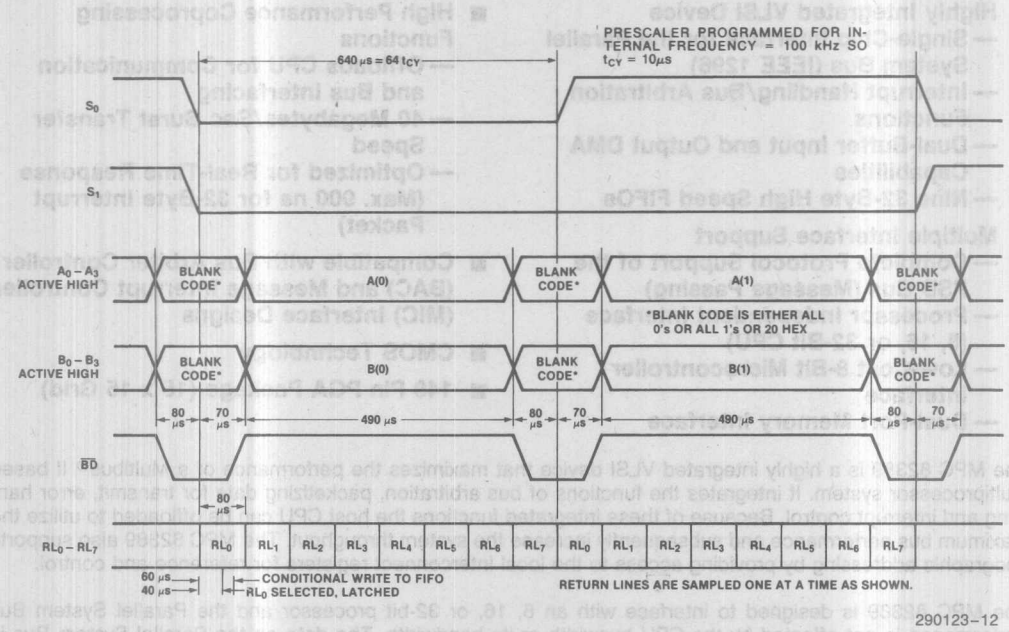
WAVEFORMS (Continued)

SCAN



WAVEFORMS (Continued)

DISPLAY



NOTE:
Shown is encoded scan left entry
S_2–S_3 are not shown but they are simply S_1 divided by 2 and 4.

82389
MESSAGE PASSING COPROCESSOR A MULTIBUS® II BUS INTERFACE CONTROLLER

- **Highly Integrated VLSI Device**
  - **— Single-Chip Interface for the Parallel System Bus (IEEE 1296)**
  - **— Interrupt Handling/Bus Arbitration Functions**
  - **— Dual-Buffer Input and Output DMA Capabilities**
  - **— Nine 32-Byte High Speed FIFOs**
- **Multiple Interface Support**
  - **— Complete Protocol Support of the PSB Bus (Message Passing)**
  - **— Processor Independent Interface (8, 16, or 32-Bit CPU)**
  - **— Low-Cost 8-Bit Microcontroller Interface**
  - **— Dual-Port Memory Interface**
- **High Performance Coprocessing Functions**
  - **— Offloads CPU for Communication and Bus Interfacing**
  - **— 40 Megabytes/Sec Burst Transfer Speed**
  - **— Optimized for Real-Time Response (Max. 900 ns for 32-Byte Interrupt Packet)**
- **Compatible with Bus Arbiter Controller (BAC) and Message Interrupt Controller (MIC) Interface Designs**
- **CMOS Technology**
- **149 Pin PGA Package (15 x 15 Grid)**

The MPC 82389 is a highly integrated VLSI device that maximizes the performance of a Multibus® II based multiprocessor system. It integrates the functions of bus arbitration, packetizing data for transmit, error handling and interrupt control. Because of these integrated functions the host CPU can be offloaded to utilize the maximum bus performance and subsequently increase the system throughput. The MPC 82389 also supports geographic addressing by providing access to the local interconnect registers for reference and control.

The MPC 82389 is designed to interface with an 8, 16, or 32-bit processor and the Parallel System Bus performance is not affected by the CPU buswidth or its bandwidth. The data on the Parallel System Bus is burst transferred at the maximum bus speed of 40 Megabytes/second regardless of CPU bus performance. Such performance is possible due to decoupling of the CPU from the Parallel System Bus.

MULTIPROCESSOR ARCHITECTURE

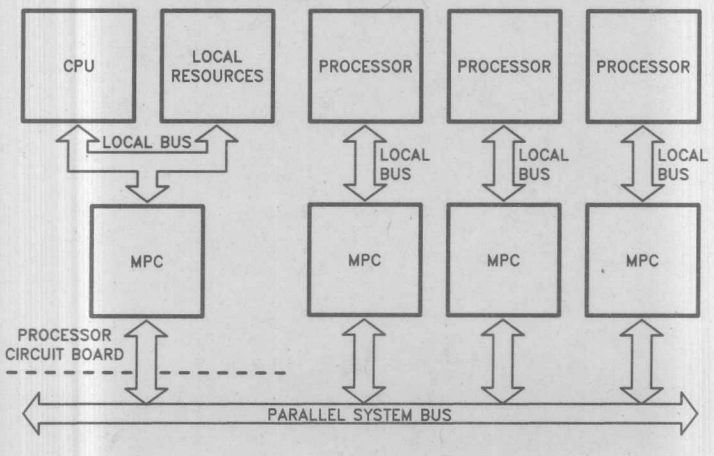


Figure 1-1

October 1987
Order Number: 290145-001

1.0 MPC 82389 INTRODUCTION

1.1 MPC Functional Overview
1.2 Major Operations of the MPC
1.3 Message Passing Protocol
1.4 Compatibility with BAC/MIC Interface Designs

2.0 MPC 82389 INTERFACES

2.1 Local Bus (Host)
2.2 PSB Bus
2.3 Interconnect Bus
2.4 Dual-Port Memory Interface
2.5 Basic Implementation of the MPC
2.6 Implementation with Dual-Port Memory

3.0 MPC 82389 INTERPROCESSOR COMMUNICATION

3.1 Communication Protcol
3.1.1 Unsolicited Message Passing
3.1.2 Solicited Message Passing
3.2 Bus Bandwidth

4.0 MPC 82389 PIN DESCRIPTION

4.1 PSB Signals
4.2 Dual-Port Memory Signals
4.3 Local Bus Signals
4.4 Interconnect Bus Signals
4.5 Power and Ground Signals

5.0 MPC 82389 MECHANICAL DATA

5.1 Pin Assignment

6.0 MPC 82389 ELECTRICAL DATA

7.0 REFERENCE DOCUMENTS

1.0 MPC 82389 INTRODUCTION

The Message Passing coprocessor 82389 is a highly integrated CMOS VLSI device to interconnect intelligent boards in a MULTIBUS II system environment. The parallel system bus of the MBII architecture definition however allows existence of intelligent and non-intelligent boards in the system.

This section of the data sheet describes the device in general including the definition of message passing protocol and the subsequent sections will contain the detailed features of the device. Please refer to the MPC User's Manual for more details.

1.1 MPC 82389 Functional Overview

The MPC 82389 is a Bus Interface Controller designed to offload the host CPU for interprocessor communication on the PSB network, and it's primary function is to support the communication protocol standard defined for the PSB bus (message passing). The device provides both the physical and data link support. By standardizing the signal interface (physical), it allows multiple vendors to offer standard add-on products for the user and at the same time it reduces costly overheads for the suppliers. The data link protocol is completely handled by the MPC 82389 including packetization after receiving data from the local interface, bus arbitration, burst transfer and error detection without the CPU intervention.

The PSB bus standard is defined for easy access and sharing of resources in a distributed processing environment. The MPC 82389 complements this standard by providing an optimized interface for the PSB bus usage at its maximum bandwidth.

The MPC 82389 also features three additional interfaces for use on a processor board.

Local Bus Interface for Host independent CPU. The CPU can be 8, 16 or 32 bits wide.

Interconnect Bus for interfacing to a low-cost microcontroller. The interconnect bus has a local address space which can be accessed by other agents on the PSB bus via the MPC.

Dual-Port Memory Interface to support an alternative communication approach which may coexist with the message passing method.

1.2 Major Operations of the MPC 82389

— Support of both unsolicited and solicited message transfers. This interprocessor communication protocol allows an intelligent agent on the PSB bus to communicate to another without any CPU intervention and at rates approaching the PSB bus bandwidth.

— Support of single cycle accesses by the host processor to memory and I/O locations resident on the PSB bus. Bus architecture, parity generation and error detection is completely handled by the MPC 82389 coprocessor.

— Support of accesses to local interconnect space by both the host processor and other agents on the PSB bus.

— Support of accesses by the host processor to interconnect location assigned to other PSB bus agents.

— Support of accesses to local, dual-port memory by other agents on the PSB bus.

1.3 Message Passing Protocol

The Multibus II architecture defines the data transfer protocol between agents on the PSB bus as Message Passing.

Message Passing allows the PSB agents to transfer variable amounts of data at rates approaching the maximum bus speed. The MPC 82389 fully supports the standardized data link protocol designed for the PSB and the entire handshaking between agents on the PSB bus is handled by the MPC 82389 without the CPU intervention.

There are two types of messages that can be transmitted from one PSB agent to another: **Unsolicited Messages** and **Solicited Messages**.

Unsolicited Messages—An unsolicited message is an intelligent interrupt also called virtual interrupt. This unsolicited message, as the name implies, is an asynchronous event to notify the receiving agent to prepare for the receipt of **Solicited Messages**. The message is in the form of a packet and it consists of information about the interrupt. By providing such intelligence the receiving agent's CPU do not have to poll for information, thus resulting in minimal latency.

Solicited Messages—The solicited messages are the actual data that are transmitted from one MPC to another. The data is once again broken into packets and these packets are transferred using the negotiation (handshaking) process which are synchronized by the MPC 82389 coprocessors.

1.4 Compatibility with BAC/MIC Interface Designs

The Bus Arbiter Controller (BAC) and Message Interrupt Contoller (MIC) were the first support components for the Parallel System Bus. The BAC implemented the full arbitration, requestor, and replier functions of the PSB. The MIC supported the transmit and receiving of minimum size unsolicited (interrupt) messages.

To ensure future compatibility with the MPC 82389 implementation, the BAC/MIC architecture was put on a module called the docket for direct incorporation onto the base-board. The PSB implementations for the MPC and the BAC/MIC docket are compatible in all respects. These implementations may coexist on the PSB bus of the same system. For the host and microcontroller interfaces, compatibility is maintained for message and interconnect space operations. It is, thus, possible to replace the BAC/MIC docket with the MPC at little impact to the board design.

Software initialization of the operating parameters for the MIC is possible through the Configuration Register to support host widths of 8, 16 or 32 bits. The MIC supports a host width of 8 bits only. Both implementations present similar software interfaces for the sending and receiving of interrupts. For details about the initialization procedures and interrupt protocols, please refer to the MPC Users's Manual, Part Number: 176526.

The MPC offers capabilities and performance far superior to the BAC/MIC implementation. Using the MPC, interrupt handling at the host interface can be improved by over an order of magnitude. Whereas the MIC can handle only minimum 4-byte interrupts, the MPC enables up to 28 bytes of data to be sent and received along with each interrupt. Further, the MPC has dedicated support for DMA based solicited message transfer.

2.0 MPC 82389 INTERFACES

The MPC 82389 features 4 interfaces: the local CPU bus for processor interface, the interconnect bus for 8-bit microcontroller interface, the Parallel System Bus interface and the dual-port memory interface.

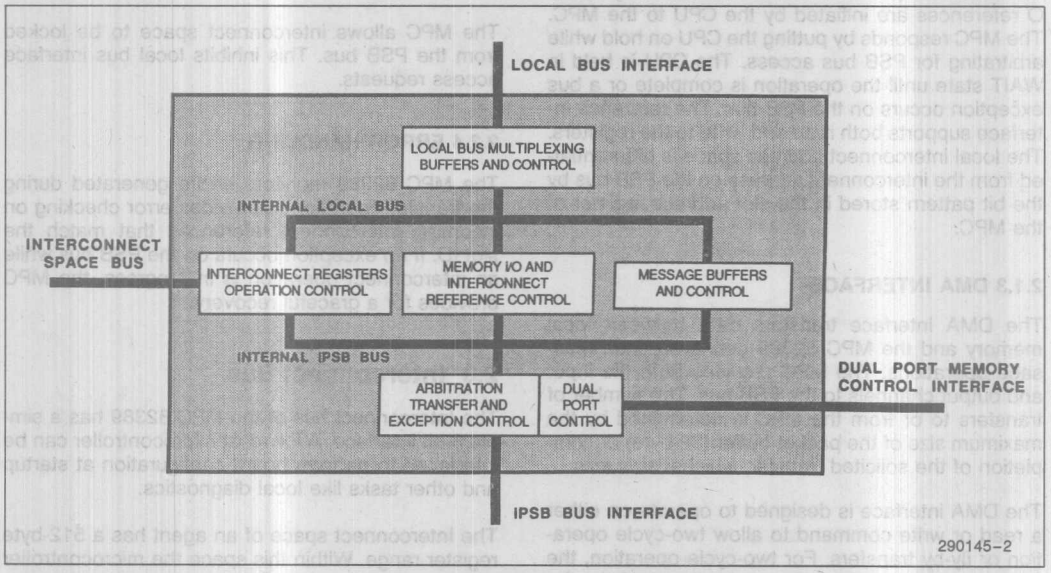


Figure 2-1. MPC Bus Interfaces

2.1 Local Bus

The local bus of the MPC 82389 is used to interface to a host processor. The CPU can be 8, 16, or 32 bits wide and the interface is processor independent.

The local bus interface supports direct references to memory, I/O and interconnect address space on the PSB bus. It also supports references to local interconnect space and the full message passing protocol. The entire local bus interface can be categorized into three sub-interfaces: register, reference and DMA.

2.1.1 REGISTER INTERFACE

The MPC 82389 local bus register interface is used for message operations and access to the interconnect space. These operations are asynchronous to the bus clock or interconnect bus operation.

2.1.2 REFERENCE INTERFACE

The MPC 82389 local bus reference interface supports direct references to memory, I/O and interconnect address space on the PSB bus. Memory and I/O references are initiated by the CPU to the MPC. The MPC responds by putting the CPU on hold while arbitrating for PSB bus access. The CPU is held in WAIT state until the operation is complete or a bus exception occurs on the PSB bus. The reference interface supports both read and write to the registers. The local interconnect address space is differentiated from the interconnect address on the PSB bus by the bit pattern stored in the slot address register of the MPC.

2.1.3 DMA INTERFACE

The DMA interface transfers data between local memory and the MPC 82389 during solicited message operations. The MPC provides both the input and output channels to the PSB bus. The number of transfers to or from the MPC is determined by the maximum size of the packet buffer (32-byte) or completion of the solicited transfer, whichever is less.

The DMA interface is designed to operate on either a read or write command to allow two-cycle operation or fly-by transfers. For two-cycle operation, the DMA uses a read operation to fetch data from the MPC and a write to put data into the MPC. Conversely, a fly-by read or write operation occurs correspondingly to memory write or read operation.

The DMA interface to the MPC performs best with aligned transfers. However, for compatibility with existing software, the MPC supports operations of arbitrary byte strings.

2.2 Parallel System Bus

The MPC 82389 provides a full 32-bit interface to the PSB bus and participates in arbitration, requestor control, replier control and error handling.

2.2.1 ARBITRATION

The MPC 82389 initials PSB bus access arbitration upon request generated inside the MPC. This request could be the result of a synchronized PSB bus reference request (memory, I/O or interconnect) or a message packet transmit request from the CPU. The PSB bus arbitration specification can be referred in the document, MPC User's Manual, Part Number: 176526.

2.2.3 REPLIER CONTROL

The MPC 82389 as a replier supports interconnect space reference and message reception. It gets into replier mode when a match is detected between the assigned slot ID and the address on the PSB bus. The interface space microcontroller is alerted of the replier mode condition. The address comparison is disabled when the MPC is the bus owner.

The MPC allows interconnect space to be locked from the PSB bus. This inhibits local bus interface access requests.

2.2.4 ERROR HANDLING

The MPC 82389 monitors errors generated during the transfer operation. It provides error checking on incoming interconnect references that match the slot ID. If an exception occurs on the PSB bus while an interconnect operation is in progress, the MPC provides for a graceful recovery.

2.3 Interconnect Bus

The Interconnect bus of the MPC 82389 has a simple 8-bit interface. A low-cost microcontroller can be interfaced to perform board configuration at startup and other tasks like local diagnostics.

The Interconnect space of an agent has a 512-byte register range. Within this space the microcontroller can store the local operating and configuration parameters associated with the agent. For example local diagnostics can be executed out of the microcontroller and the results posted in the Interconnect space. IEEE 1296 specifications require the first record in the Interconnect space to contain the board ID and Intel recommends other record types. Refer to Interconnect Interface Specification, Part Number: 149299.

The MPC 82389 provides the path to access the local Interconnect space. The references supported are:

1. CPU local bus to the local Interconnect space
2. CPU local bus to the Interconnect space of another agent on the PSB bus
3. From the PSB bus to the local Interconnect space

The local Interconnect accesses are identified as slot IDs 24–31 and the Interconnect accesses on the PSB bus are mapped as slot IDs 0–23.

The MPC participates in PSB bus handshake protocol, parity generation and checking and agent error generation for local Interconnect accesses from the PSB bus.

The Interconnect microcontroller is the master device on the bus and all other devices including the MPC are slaves.

2.4 Dual-Port Memory Interface

The MPC 82389 supports the dual-port memory interface for those designs that must coexist with the memory passing architecture.

The dual-port services supported are: Address recognition, PSB bus replier handshake, error checking, and bus parity generation and checking. A useful recovery mechanism is provided by the MPC should a bus exception error occur while a dual-port memory access is in progress. Although the MPC 82389 provides bus parity check it is the responsibility of the memory controller to generate and check data parity.

2.5 Basic Implementation with the MPC 82389

Figure 2-2 shows a basic implementation of the MPC 82389. Included in this implementation is the interconnect interface to a microcontroller, the CPU interface and the PSB bus interface.

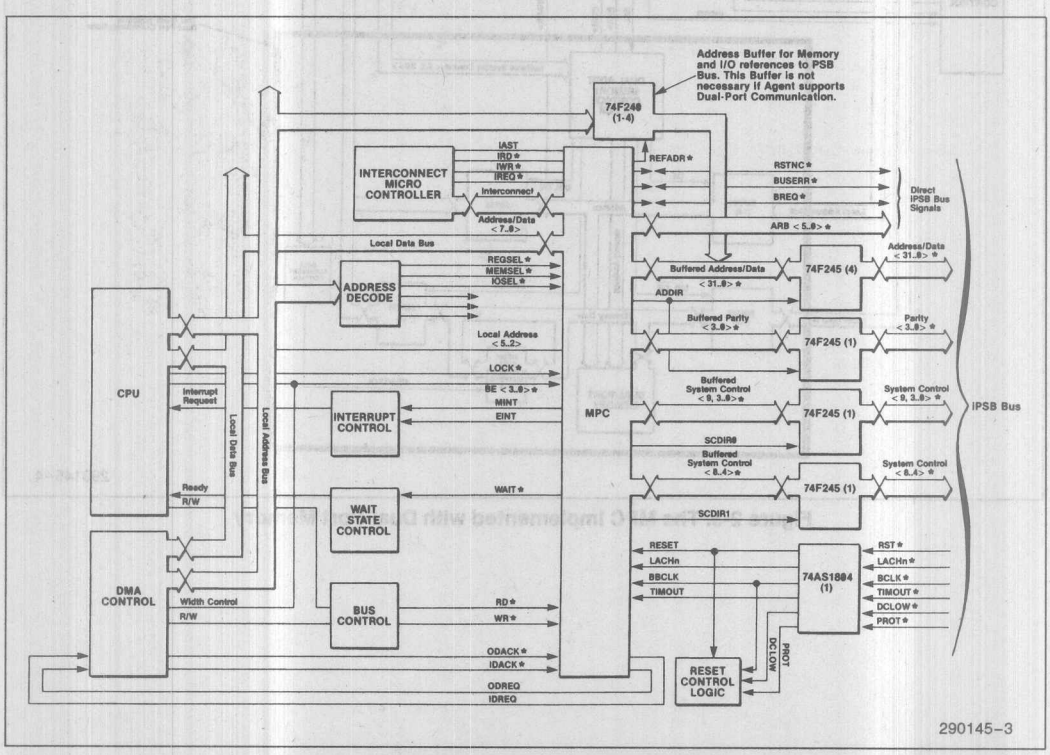


Figure 2-2. MPC Implementation to Support References

2.6 Implementation with Dual-Port Memory Interface

Figure 2-3 shows the logic required to implement dual-port operations.

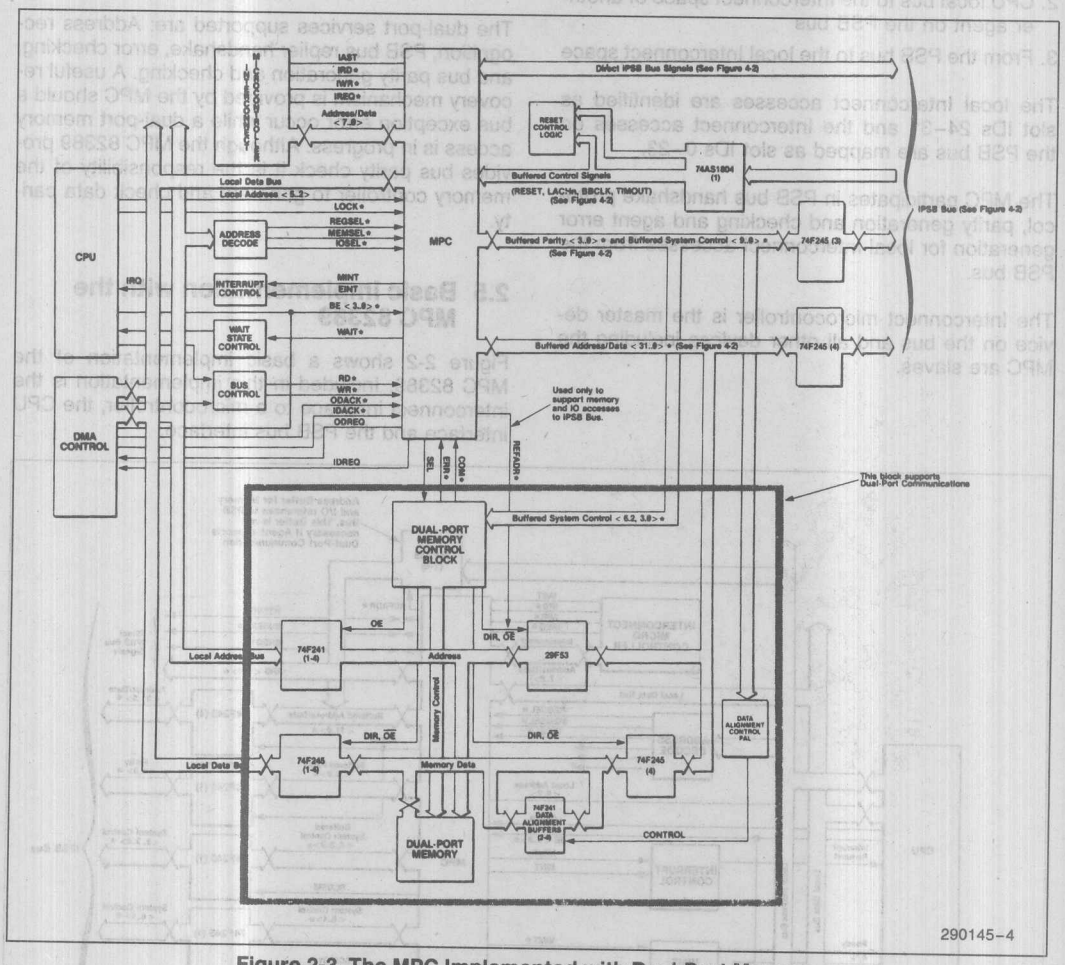


Figure 2-3. The MPC Implemented with Dual-Port Memory

3.0 MPC 82389 INTERPROCESSOR COMMUNICATION

A MULTIBUS II system can have up to 20 boards in the slot backplane. Slot zero must have a Central Services Module (CSM) which provides bus initialization and clocking. The remaining 19 slots can have a mix of intelligent and non-intelligent boards. The intelligent boards will typically communicate over the high speed PSB bus. Any processor based board may contain an MPC 82389 for high speed communication and the MPC is designed to support the system performance and the data transfer speed of the PSB bus. The MPC has an optimized PSB bus interface.

Message Passing over the PSB bus is completely handled by the MPC 82389 coprocessor without CPU intervention. The messages (data) are packetized in blocks of 32 bytes and burst transferred over the PSB bus.

The decoupling is achieved by using very high speed FIFOs of the MPC 82389. Nine 32-byte FIFOs are used in the MPC 82389. Five of these FIFOs are used for setting up the unsolicited messages (interrupts). One is used for output set up and the other four for input set up of up to four unsolicited messages. For data transmission (solicited messages) over the PSB, the MPC 82389 has two dedicated solicited output and input channels. With each channel, dual 32-byte FIFOs are used to pipeline data during output and input operation.

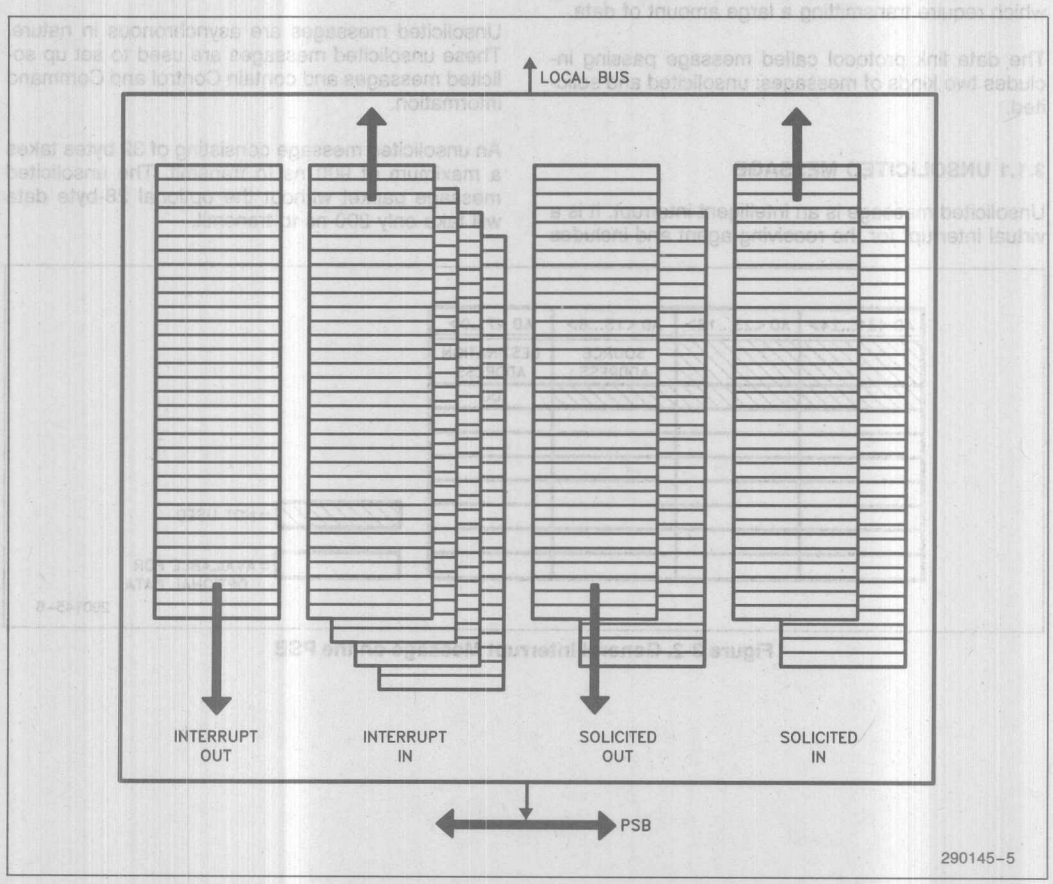


Figure 3-1. The MPC Uses Nine 32 Byte FIFOs to Couple the Local and System Buses

3.1 Communication Protocol

Any device with an interface to the PSB bus is termed as an agent. The agents communicate over the PSB bus and they completely offload the CPU for other tasks. Each agent is assigned an 8-bit address for identification.

The mechanism used for interprocessor communication over the PSB entails a standardized data link protocol, and a dedicated address space. The information which is packetized is transmitted to a dedicated address space instead of the target memory. This addressing scheme serves to decouple the CPU from the PSB bus. Packetization serves to limit the time an agent has an access over the bus, thus protecting against hogging of the bus by agents which require transmitting a large amount of data.

The data link protocol called message passing includes two kinds of messages: unsolicited and solicited.

3.1.1 UNSOLICITED MESSAGE

Unsolicited message is an intelligent interrupt. It is a virtual interrupt for the receiving agent and includes all the information required to service the interrupt. It takes only 900 ns to send an unsolicited message over the PSB bus, and since the receiving agent's processor does not have to poll for servicing the interrupt, this mechanism is fast and efficient.

The unsolicited message is sent as a packet consisting of up to 32 bytes, as shown in Figure 3-2. The address field is 8 bits long. Up to 255 agents can be addressed uniquely while one address is used for broadcast function.

The general format for unsolicited messages varies depending on the CPU bus width. There are other variations depending on whether the CPU is receiving or transmitting the message and the type of unsolicited message.

Unsolicited messages are asynchronous in nature. These unsolicited messages are used to set up solicited messages and contain Control and Command information.

An unsolicited message consisting of 32 bytes takes a maximum of 900 ns to transmit. The unsolicited message packet without the optional 28-byte data will take only 200 ns to transmit.

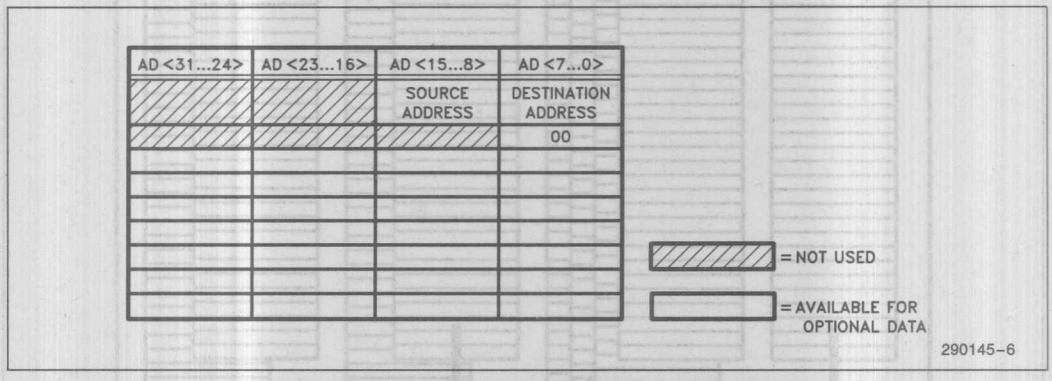


Figure 3-2. General Interrupt Message on the PSB

AD<31..24>	AD<23..16>	AD<15..8>	AD<7..0>
		Source Add. Byte 1	Dest. Add. Byte 0
		ID Byte 3	Type Byte 2
Data, Byte 7	Data, Byte 6	Data, Byte 5	Data, Byte 4
Data, Byte 11	Data, Byte 10	Data, Byte 9	Data, Byte 8
Data, Byte 15	Data, Byte 14	Data, Byte 13	Data, Byte 12
Data, Byte 19	Data, Byte 18	Data, Byte 17	Data, Byte 16
Data, Byte 23	Data, Byte 22	Data, Byte 21	Data, Byte 20
Data, Byte 27	Data, Byte 26	Data, Byte 25	Data, Byte 24
Data, Byte 31	Data, Byte 30	Data, Byte 29	Data, Byte 28
Data, Byte 35 Solic. only	Data, Byte 34 Solic. only	Data, Byte 33 Solic. only	Data, Byte 32 Solic. only

Figure 3-3. General Message Packet Format on the PSB Bus

3.1.2 SOLICITED MESSAGE

Solicited message consists of the actual data to be transferred from one agent to another over the PSB bus. The data is packetized in blocks of 32 bytes for transfer. Up to 16 Megabytes of data may be transferred in a single solicited message.

The transfer of data is negotiated between the transmitting and receiving agents via unsolicited messages. By using the acknowledge response method through the unsolicited messages, the agents complete the transfer of data.

A solicited message contains one or more data packets. The packetization of solicited messages is handled by the MPC. The padding of header information at the transmitting end and the stripping of this information out of the packet is solely the MPC responsibility. The local CPU simply fills or empties the FIFO over the local bus. The MPC also handles the last packet fillers to maintain the 32-byte data packet format. If necessary, during output the bytes are padded and during input the padded bytes are stripped by the MPC.

For programming details consult MPC User's Manual, Part Number: 176526.

3.2 Bus Bandwidth

The advantages of decoupling the buses can be summarized in Figure 3-4. The effective speed performance numbers are also listed. The first advantage is that no resource is held in wait states while arbitration for another resource is occurring. The second advantage is that each transfer can occur at the full bandwidth of the associated bus.

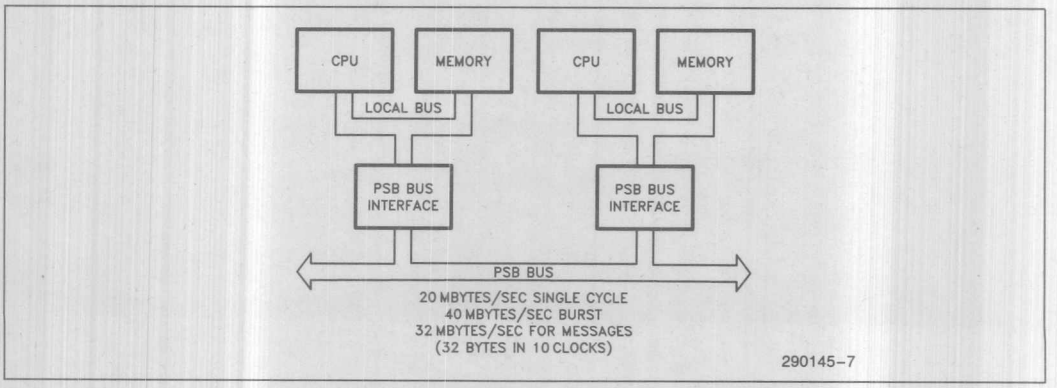


Figure 3-4. Message Passing Performance Example

4.0 MPC 82389 PIN DESCRIPTION

The MPC 82389 is packaged in a 149 pin package. The signals for the device are functionally divided by their associated interfaces as shown in Figure 4-1.

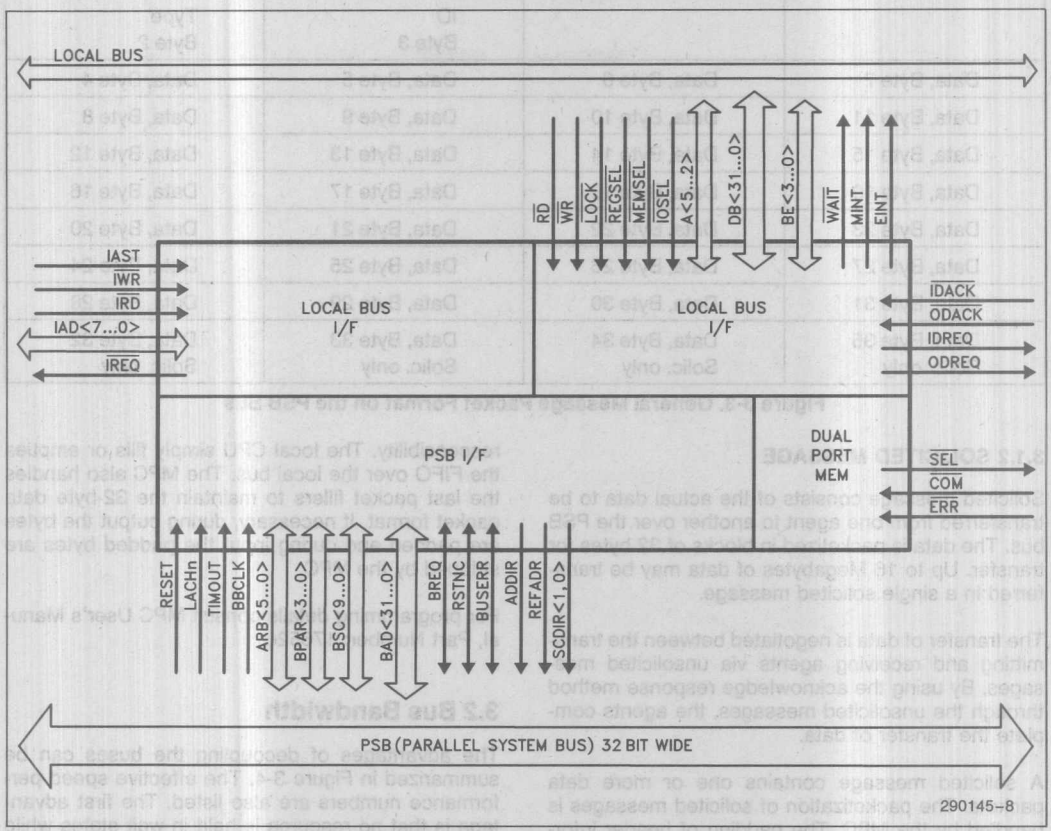


Figure 4-1. MPC Functional Blocks

4.1 PSB Bus Signals

This section describes each of the PSB bus signals that interface with the MPC. For complete descriptions of these signals, see the MULTIBUS II Architecture Specification, Part Number: 146077.

The PSB bus signals interfaced by the MPC 82389 fall into five groups, depending on function:

- Arbitration Operation Signal Group
- Address/Data Bus Signal Group
- System Control Signal Group
- Central Control Signal Group
- Exception Operation Signal Group

Unless otherwise stated, all PSB bus signals are synchronous to the bus clock.

4.1.1 ARBITRATION OPERATION SIGNAL GROUP

The MPC 82389 interfaces directly with the Arbitration Operation Signal Group of the PSB bus. These are all high-current drive, open-collector signals. Below is a description of each signal.

$\overline{\text{BREQ}}$ (Bus Request)
$\overline{\text{BREQ}}$ is a bidirectional open-collector signal that connects directly to the PSB bus. As an input to the MPC, it indicates that agents are awaiting access to the bus. In fair access mode, this inhibits the MPC from activating its own request. As an output, the MPC asserts $\overline{\text{BREQ}}$ to request access to the PSB bus.

$\overline{\text{ARB}}$<5..0> (Arbitration)
$\overline{\text{ARB}}$<5..0> are the arbitration signals for the PSB bus. At the MPC interface, these are bidirectional, open-collector signals that connect directly to the PSB bus. $\overline{\text{ARB}}$<5..0> are used during normal operation to identify the mode and arbitration priority of an agent during an arbitration cycle to facilitate the arbitration process. During system initialization (while reset is active), the Central Services Module (CSM) drives these signals to initialize slot and arbitration IDs.

4.1.2 ADDRESS/DATA BUS SIGNAL GROUP

This signal group includes a 32-bit multiplexed address/data path that interfaces to the PSB address/data bus. The MPC also includes the byte parity signals present on the PSB bus $\overline{\text{BPAR}}$<3..0>. All signals in this group interface with the PSB bus through bus tranceivers. For the MPC, this signal group also includes signals to control these bus transceivers (ADDIR and $\overline{\text{REFADR}}$). These signals are described next.

$\overline{\text{BAD}}$<31..0> (Buffered Address/Data)
$\overline{\text{BAD}}$<31..0> are the 32 buffered, multiplexed address/data signals that are bidirectional and provide the interface to the PSB address/data bus. At the MPC, these lines should be connected to the equivalent PSB bus $\overline{\text{AD}}$ signals using 74F245 or equivalent transceivers.

$\overline{\text{BPAR}}$<3..0> (Buffered Parity)
$\overline{\text{BPAR}}$ are four signals that provide parity for the 4 bytes of the $\overline{\text{BAD}}$ bus. These bidirectional lines connect to the PSB bus $\overline{\text{PAR}}$<3..0> signals through a 74F245 or equivalent transceiver. These signals are used to receive byte parity for incoming operations and to drive byte parity for outgoing operations.

ADDIR (Address/Data Direction)
ADDIR is an output that provides direction control over the transceivers driving and receiving $\overline{\text{BAD}}$<31..0> and $\overline{\text{BPAR}}$<3..0>. In the high state, this signal causes the transceivers to place address/data information along with parity onto the PSB bus. In the low state, this signal causes address/data information and parity to be received from the PSB bus.

$\overline{\text{REFADR}}$ (Reference Address Enable)
$\overline{\text{REFADR}}$ is an output used to enable external address buffers. Asserting this signal places address information from the local bus onto $\overline{\text{BAD}}$. The address path enabled by this signal is used for memory and I/O references to the PSB bus and is not used during message passing or for references to interconnect space on the PSB bus.

4.1.3 SYSTEM CONTROL SIGNAL GROUP

The MPC provides signals that are used to interface to the System Control Signal Group of the PSB bus. These signals are described next.

$\overline{\text{BSC}}$<9..0> (Buffered System Control)
$\overline{\text{BSC}}$<9..0> is a group of ten bidirectional signals that interface to the System Control Signal Group of the PSB bus through 74F245 or equivalent transceivers. Direction control of the transceivers is provided by SCDIR<1, 0> (discussed next). Agents on the PSB bus use the System Control Signal Group to define commands or report status, depending on the phase of the operation. See the MULTIBUS II Architecture Specification for more information on these signals.

SCDIR<1, 0> (System Control Direction)
SCDIR<1, 0> are output signals that provide direction control of the 74F245 transceivers driving and receiving $\overline{\text{BSC}}$<9..0>. SCDIR0 provides control for $\overline{\text{BSC}}$<9, 3..0>, while SCDIR1 provides control for

$\overline{BSC}$<8..4>. When either signal is high, the corresponding five bits of the $\overline{BSC}$ signal group are driven onto the PSB bus. When either signal is low, the corresponding five bits on the PSB bus are driven onto the $\overline{BSC}$ signal group.

4.1.4 CENTRAL CONTROL SIGNAL GROUP

The MPC provides several signals that interface directly or through transceivers to the Central Control signal group of the PSB bus. These signals are described next.

BBCLK (Buffered Bus Clock)
BBCLK is buffered from the PSB bus $\overline{BCLK}$ signal. This signal should be connected to $\overline{BCLK}$ using a 74AS1804 or equivalent inverting buffer. This clock is used for all synchronous internal MPC timing.

LACHn (Latch n)
LACHn is an input signal used during initialization of slot and arbitration IDs (where "n" is the slot number). When the RESET signal is active, LACHn asserted indicates to an agent that a slot or arbitration ID is available and should be latched. LACHn is an active high input and should be connected to the $\overline{LACHn}$ signal on the PSB bus with a 74AS1804 or equivalent inverting buffer.

RESET
RESET is an input that, when asserted, places the MPC in a known state. Only the parts of the MPC involved with initialization of slot and arbitration IDs remain unaffected. RESET is an active high input and should be connected to the $\overline{RST}$ signal on the PSB bus with a 74AS1804 or equivalent inverting buffer.

Reset Condition
Table 4-1 summarizes the states of the signals while the RESET signal is active.

Table 4-1. Signal State During Reset

Signal	Reset State
$\overline{BREQ}$, $\overline{ARB}$<5..0>	Z
$\overline{BAD}$<31..0>	Z
ADDIR	L
$\overline{REFADR}$	H
$\overline{BSC}$<9..0>	Z
SCDIR<1, 0>	L
$\overline{BUSERR}$	Z(H)
$\overline{RSTNC}$	L
$\overline{SEL}$	H
D<31..0>	Z
$\overline{WAIT}$	H
MINT, EINT	L
ODREQ, IDREQ	L

NOTES:
H — Electrical High State
L — Electrical Low State
Z — High Impedence

$\overline{RSTNC}$ (Reset Not Complete)
$\overline{RSTNC}$ is a bidirectional, open-collector signal with high-current drive. It connects directly to the PSB bus. As an input, $\overline{RSTNC}$ inhibits the MPC from initiating PSB bus operations. As an output, the MPC asserts $\overline{RSTNC}$ to prevent PSB bus operation until the agent is finished with initialization. The MPC asserts $\overline{RSTNC}$ whenever the $\overline{RST}$ signal is asserted by the Central Services Module (CSM). After the CSM deasserts $\overline{RST}$ and initialization of the local agent is complete, the interconnect microcontroller writes to a register within the MPC. The MPC then deasserts $\overline{RSTNC}$.

4.1.5 EXCEPTION OPERATION SIGNAL GROUP

The MPC interfaces with both signals of the Exception Operation Signal Group (part of the PSB bus), as described below.

$\overline{BUSERR}$ (Bus Error)
$\overline{BUSERR}$ is a bidirectional, open-collector signal with high-current drive. It connects directly to the PSB bus. As an input, the MPC uses this signal to detect bus errors signaled by other agents. As an output, the MPC uses $\overline{BUSERR}$ to indicate parity errors detected on either the $\overline{BAD}$ or $\overline{BSC}$ signals and to indicate handshake protocol violations detected on the $\overline{BSC}$ signals.

TIMOUT (Time-Out)
TIMOUT is an input from the PSB bus used to detect a time-out condition signaled by the CSM. TIMOUT is an active high input to the MPC and must be connected to the $\overline{TIMOUT}$ signal of the PSB bus through a 74AS1804 or equivalent inverter buffer.

4.2 Dual-Port Memory Control Signals

The MPC provides the following signals to support dual-port memory.

$\overline{SEL}$ (Select)
The MPC asserts $\overline{SEL}$ to indicate that a dual-port memory access is in progress. The assertion of $\overline{SEL}$ initiates the dual-port operation and during memory reads, can be used to enable the dual-port data buffers onto the $\overline{BAD}$ bus. When the MPC completes the PSB bus handshake on the PSB bus, or if the MPC detects an exception, it deasserts $\overline{SEL}$.

$\overline{COM}$ (Complete)
$\overline{COM}$ is an input to the MPC. The dual-port memory controller asserts $\overline{COM}$ to indicate it is ready to complete dual-port access. $\overline{COM}$ is assumed to be synchronous to the bus clock. The MPC asserts the Replier Ready ($\overline{SC4}$) signal on the PSB bus on the bus clock after the memory controller has asserted $\overline{COM}$. The memory controller cannot deassert $\overline{COM}$ until the end-of-transfer (EOT) handshake is complete on the PSB bus. This requires that the memory controller monitor the PSB bus for the EOT handshake.

$\overline{ERR}$ (Error)
$\overline{ERR}$ is asserted by the dual-port memory controller to signal a memory data parity error. $\overline{ERR}$ must be stable (high or low) whenever $\overline{COM}$ is asserted. The MPC responds to this signal by completing the replier handshake on the PSB bus using a "data error" agent error code. This signal may be asynchronous to the bus clock since it is qualified by the $\overline{COM}$ signal.

4.3 Local Bus Signals

The MPC provides five signal groups that together interface to the CPU's local bus. These local bus signal groups are:

- data
- address and select
- transfer control
- interrupt
- DMA control

All local bus signals are assumed to be asynchronous to the bus clock.

4.3.1 DATA BUS

The local data bus (D<31..0>) is a bidirectional group of signals that transfers data between the host CPU, DMA controller, or memory and the MPC. Although this is a 32-bit interface, the MPC provides control to allow operation with processors using 8-, 16-, or 32-bit data buses.

Not all processors use the same byte order when performing multiple data byte operations. For example, for a 16-bit write to memory, one processor may carry the least-significant byte on local bus bits D<7..0> and the most-significant byte on bits D<15..8>, while another processor may carry the least-significant byte on bits D<15..8> and the most-significant byte on bits D<7..0>. For a given agent, be sure to implement the processor interface to maintain consistent byte addressability with all other agents in the system.

4.3.2 ADDRESS AND SELECT SIGNALS

The address and status signals identify all MPC operations over the local bus.

A<5..2> (Address)
The address inputs select MPC registers for message and interconnect space operations. A1 and A0 are omitted to provide a consistent register address for all data bus width options. These signals are qualified by commands in the MPC (for example, $\overline{RD}$ or $\overline{WR}$, defined in section 4.1.3.3). To the MPC, the state of A<5..2> must be stable within the specified setup and hold window. The address values defined by A<5..2>, and required to access MPC registers, are provided in, "Programming the Host Interface" of the MPC User's Manual, Part Number: 176526.

$\overline{BE}$<3..0> (Byte Enable)
These input signals identify valid bytes for memory and I/O reference operations and also provide data path control for register and DMA operations. The assertion of a byte enable signal validates a particular byte on the data bus. Signals $\overline{BE}$<3..0> correspond to data bytes 3 through 0 on the data bus (where byte 3 is D<31..24>). Only combinations supported by the PSB bus specification are valid. Valid combinations are summarized in Table 4-2. Values not shown in the table are illegal and will result in unpredictable operation. These signals are qualified by commands (for example, $\overline{RD}$ or $\overline{WR}$) in the MPC and must be stable within the specified setup and hold window.

Operation with 32-bit local buses requires that all byte enable and data signals are used. For 16-bit local buses, $\overline{BE2}$ and $\overline{BE33}$ are deasserted, $\overline{BE1}$ and $\overline{BE2}$ are used to indicate which of the two bytes will contain valid data, and only D<15..0> are used. For 8-bit local bus operations, $\overline{BE3}$ is asserted, $\overline{BE2}$ is deasserted, and $\overline{BE1}$ and $\overline{BE0}$ are used to select which byte of the PSB bus will carry the valid data byte. This mode uses only D<7..0> (on the local bus). Note that during all read operations, the MPC drives D<31..0>.

Table 4-2. Valid Byte Enable Combinations

				Local Bus				PSB Bus†			
$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	D31–D24	D23–D16	D15–D8	D7–D0	AD31–AD24	AD23–AD16	AD15–AD8	AD7–AD0
L	L	L	L	V3	V2	V1	V0	V3	V2	V1	V0
L	L	L	H	V3	V2	V1	x	V3	V2	V1	x
H	L	L	L	x	V2	V1	V0	x	V2	V1	V0
L	L	H	H	V3	V1	x	x	x	x	V3	V2
H	L	L	H	x	V2	V1	x	x	V2	V1	x
H	H	L	L	x	x	V1	V0	x	x	V1	V0
L	H	H	H	V3	x	x	x	x	x	V3	x
H	L	H	H	x	V2	x	x	x	x	x	V2
H	H	L	H	x	x	V1	x	x	x	V1	x
H	H	H	L	x	x	x	V0	x	x	x	V0
L	H	L	H	x	x	x	V0	x	x	V0	x
L	H	H	L	x	x	x	V0	x	x	x	V0

NOTES:
L — Electrical low state (active)
H — Electrical high state (inactive)
Vx — Valid data bytes
x — Active bytes with undefined data
† — For this PSB bus, these combinations apply to reference operations, not message space operations

$\overline{MEMSEL}$ (Memory Select)
This MPC input signal, when asserted, indicates to the MPC that the current operation is a memory reference to the PSB bus. It is qualified by the assertion of $\overline{RD}$ or $\overline{WR}$ (defined in section 4.3.3). The state of $\overline{MEMSEL}$ be must stable within the defined set-up and hold window. Additionally, for $\overline{MEMSEL}$ to be valid, the signals $\overline{IOSEL}$, $\overline{REGSEL}$, $\overline{IDACK}$, and $\overline{ODACK}$ must not be active during the same setup and hold window. ($\overline{IDACK}$ and $\overline{ODACK}$ are defined later in section 4.3.5.)

$\overline{IOSEL}$ (I/O Select)
This input signal, when asserted, indicates to the MPC that the current operation is an I/O reference to the PSB bus. It is qualified by the assertion of $\overline{RD}$ or $\overline{WR}$ (defined in section 4.3.3). The state of $\overline{IOSEL}$ must be stable within the defined setup and hold window. Additionally, for $\overline{IOSEL}$ to be valid, the signals $\overline{MEMSEL}$, $\overline{REGSEL}$, $\overline{IDACK}$, and $\overline{ODACK}$ must not be active during the same setup and hold window. ($\overline{IDACK}$ and $\overline{ODACK}$ are defined later in section 4.3.5.)

$\overline{REGSEL}$ (Register Select)

This input signal, when asserted, identifies an operation as an MPC-register access. The host CPU asserts $\overline{REGSEL}$ to set up the MPC for message or interconnect space operations and these are mapped as register operations. $\overline{REGSEL}$ is qualified by the assertion of $\overline{RD}$ or $\overline{WR}$ (defined in section 4.3.3). The state of $\overline{REGSEL}$ must be stable within the defined setup and hold window. Additionally, for $\overline{REGSEL}$ to be valid, the signals $\overline{MEMSEL}$, $\overline{IOSEL}$, $\overline{IDACK}$, and $\overline{ODACK}$ must not be active during the same setup and hold window. ($\overline{IDACK}$ and $\overline{ODACK}$ are defined later in section 4.3.5).

$\overline{LOCK}$
This input signal allows back-to-back operations to be performed on the PSB bus or to local interconnect space. When $\overline{LOCK}$ is asserted, any resource accessed by the operation (PSB bus or local interconnect space) is locked until $\overline{LOCK}$ is deasserted.

4.3.3 TRANSFER CONTROL SIGNALS

Transfer control to the MPC over the local bus is provided by two command signals (Read and Write) and a wait signal. This handshake provides fully interlocked (two-sided handshake) operation.

$\overline{RD}$ (Read)
This input signal, when asserted, generally initiates a read operation. The CPU asserts $\overline{RD}$ to initiate read operations of MPC registers. The CPU also asserts $\overline{RD}$ to initiate read operations of I/O and memory locations present on the PSB bus. The DMA controller asserts $\overline{RD}$ to qualify DMA cycles. In this last case, the MPC does not interpret $\overline{RD}$ as an indicator of the data transfer direction, but only to qualify the DMA acknowledge signal (see definitions for $\overline{ODACK}$ and $\overline{IDACK}$). $\overline{RD}$ must transition cleanly, since it is used to latch other signals that define the parameters of the operation.

$\overline{WR}$ (Write)
This input signal, when asserted, generally initiates a write operation. The CPU asserts $\overline{WR}$ to initiate write operations of MPC registers. The CPU also asserts $\overline{WR}$ to initiate write operations of I/O and memory locations present on the PSB bus. The DMA controller asserts $\overline{WR}$ to qualify DMA cycles. In this last case, the MPC does not interpret $\overline{WR}$ as an indicator of the data transfer direction, but only to qualify the DMA acknowledge signal (see definitions for $\overline{ODACK}$ and $\overline{IDACK}$). $\overline{WR}$ must transition cleanly, since it is used to qualify other signals that define the parameters of the operation.

$\overline{WAIT}$
$\overline{WAIT}$ is an MPC output signal used to delay (or suspend) a local bus operation during an access to an asynchronous resource via the MPC. The MPC asserts $\overline{WAIT}$ to the local CPU for memory, I/O, and interconnect accesses to the PSB bus; and for local interconnect accesses. $\overline{WAIT}$, when asserted, allows time for the accessed resource to become available. The MPC asserts $\overline{WAIT}$ after the CPU has asserted the command signal ($\overline{RD}$ or $\overline{WR}$). On the PSB bus, the MPC deasserts $\overline{WAIT}$ after either the PSB bus EOT handshake or an exception has occurred. For accesses to local interconnect space, the MPC deasserts $\overline{WAIT}$ after the interconnect operation is complete.

4.3.4 INTERRUPT SIGNALS

Interrupt signals are used to inform the host CPU that the MPC requires service. The MPC provides two signals: one for message operations and one for reference errors.

MINT (Message Interrupt)
The MPC asserts this output signal for all message-related signaling to the host CPU. This includes the arrival of an unsolicited message, an available transmit FIFO buffer, the completion of a solicited transfer, and an error on message transfer.

EINT (Error Interrupt)
The MPC asserts this output signal to the CPU to indicate errors related to memory, I/O, or interconnect space operations (i.e., all except message operations). Internal registers in the MPC provide details of the error via interconnect space.

4.3.5 DMA CONTROL SIGNALS

The MPC provides several DMA control signals to support an external DMA controller. A DMA controller is required to support solicited message operations.

ODREQ (Output Channel DMA Request)
ODREQ is an output signal the MPC asserts to enable DMA transfer of data to the MPC (i.e., output to the PSB bus). This signal behaves as a normal DMA request line. For a solicited message output operation, the MPC asserts ODREQ when a solicited output packet buffer is empty and as long as the MPC is in the transfer phase (the Buffer Request unsolicited message has been sent). The DMA controller responds by performing DMA transfers to the MPC for transfer to the receiving agent.

IDREQ (Input Channel DMA Request)
The MPC asserts this output signal to enable DMA transfer of data from the MPC (i.e., input from the PSB bus). This signal behaves as a normal DMA request line. For a solicited message input operation, the MPC asserts IDREQ after a solicited input packet buffer is full and as long as the MPC is in the transfer phase. The DMA controller responds by performing DMA transfer from the MPC. IDREQ remains asserted until the packet is transferred to memory.

$\overline{ODACK}$ (Output Channel DMA Acknowledge)
$\overline{ODACK}$ is an input signal asserted by the DMA controller in response to the assertion of ODREQ by the MPC. The DMA controller asserts $\overline{ODACK}$ to set up the MPC for the DMA transfer from local memory (or the controller) to the MPC. The assertion of $\overline{ODACK}$ is qualified by the assertion of $\overline{RD}$ or $\overline{WR}$ by the MPC. The direction of data transfer with respect to the MPC is controlled by the request signal (IDREQ or ODREQ) and the acknowledge signal ($\overline{IDACK}$ or $\overline{ODACK}$). The command signal ($\overline{RD}$ or $\overline{WR}$) only qualifies the acknowledge signal ($\overline{IDACK}$ or $\overline{ODACK}$). The state of $\overline{ODACK}$ must be stable within the defined setup and hold window. Additionally, for $\overline{ODACK}$ to be valid, the signals $\overline{MEMSEL}$, IOSEL, $\overline{REGSEL}$, and $\overline{IDACK}$ must not be active during the same setup and hold window.

$\overline{IDACK}$ (Input Channel DMA Acknowledge)
$\overline{IDACK}$ is an input signal asserted by the DMA controller in response to the assertion of IDREQ by the MPC. The DMA controller asserts $\overline{IDACK}$ to set up the MPC for the DMA transfer from the MPC to the DMA controller (or local memory). The assertion of $\overline{IDACK}$ is qualified by the assertion of $\overline{RD}$ or $\overline{WR}$ by the DMA controller. The direction of data transfer with respect to the MPC is controlled by the request signal (IDREQ or ODREQ) and the acknowledge signal ($\overline{IDACK}$ or $\overline{ODACK}$). The command signal ($\overline{RD}$ or $\overline{WR}$) only qualifies the acknowledge signal. The state of $\overline{IDACK}$ must be stable within the de-

fined setup and hold window. Additionally, for $\overline{IDACK}$ to be valid, the signals $\overline{MEMSEL}$, $\overline{IOSEL}$, $\overline{REGSEL}$, and $\overline{ODACK}$ must not be active during the same setup and hold window.

4.4 Interconnect Bus Signals

The interconnect bus signals provide a simple interface to a microcontroller for implementation of interconnect space. All interconnect bus signals are asynchronous to the bus clock and to the local bus signals.

IAD<7..0> (Interconnect Address/Data)
IAD<7..0> is an 8-bit, bidirectional, multiplexed address and data bus intended to interface directly to a microcontroller. In addition to the MPC, other interconnect registers can be connected to this bus.

$\overline{IREQ}$ (Interconnect Request)
The MPC asserts this output signal when an interconnect operation has been requested from either the local bus or the PSB bus. The MPC deasserts $\overline{IREQ}$ after the microcontroller has written to the Interconnect Reference Arbitration register.

IAST (Interconnect Address Strobe)
IAST is an input signal from the microcontroller and, when asserted, indicates that a valid address is on the interconnect bus. IAST may be directly connected to the ALE (Address Latch Enable or equivalent) output of most microcontrollers. IAST must provide clean transitions.

$\overline{IRD}$ (Interconnect Bus Read)
$\overline{IRD}$ is an input signal. The microcontroller asserts $\overline{IRD}$ to perform a read operation to one of the MPC interconnect interface registers. $\overline{IRD}$ must provide clean transitions. When $\overline{IRD}$ is asserted in conjunction with the $\overline{IWR}$ signal, all MPC outputs are disabled.

$\overline{IWR}$ (Interconnect Write)
$\overline{IWR}$ is an input signal. The microcontroller asserts $\overline{IWR}$ to perform a write operation to one of the MPC interconnect interface registers. $\overline{IWR}$ must provide clean transitions. When $\overline{IWR}$ is asserted in conjunction with the $\overline{IRD}$ signal, all MPC outputs are disabled.

4.5 Power and Ground Signals

The MPC requires supply voltage and ground connections at the pin numbers listed below.

V_{CC}	Ground
D4	J3
M4	N4
N8	N6
M12	N9
D12	N11
C7	N13
	K13
	F13
	C12
	D8
	C5
	C3

5.0 MPC 82389 MECHANICAL DATA

The MPC 82389 is packaged in a 149 lead pin grid array. The square package has a 15 x 15 grid layout with the outer 3 rows used along each edge.

5.1 Pin Assignment

The MPC 82389 pinout as viewed from the top side of the component is shown in Figure 5-1. When viewed from the pin side, the component pin layout is shown in Figure 5-2.

To reduce possible noise problems on the board, V_{CC} and V_{SS} must be connected to multiple supplies. The board should be laid out with V_{CC} and Ground planes for power distribution and the components V_{CC} and V_{SS} must be connected to the appropriate power plane.

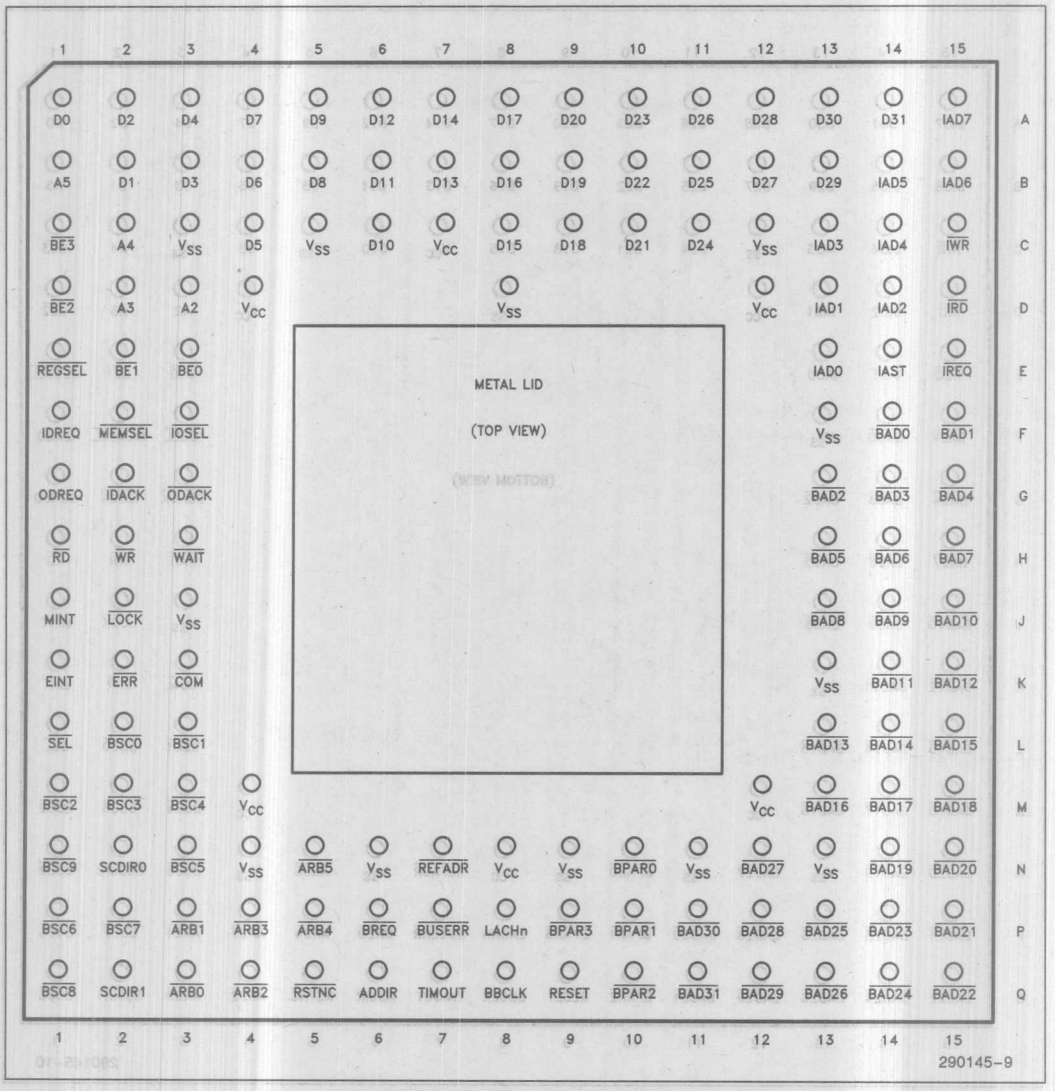


Figure 5-1. MPC 82389 Pinout—View from Top Side

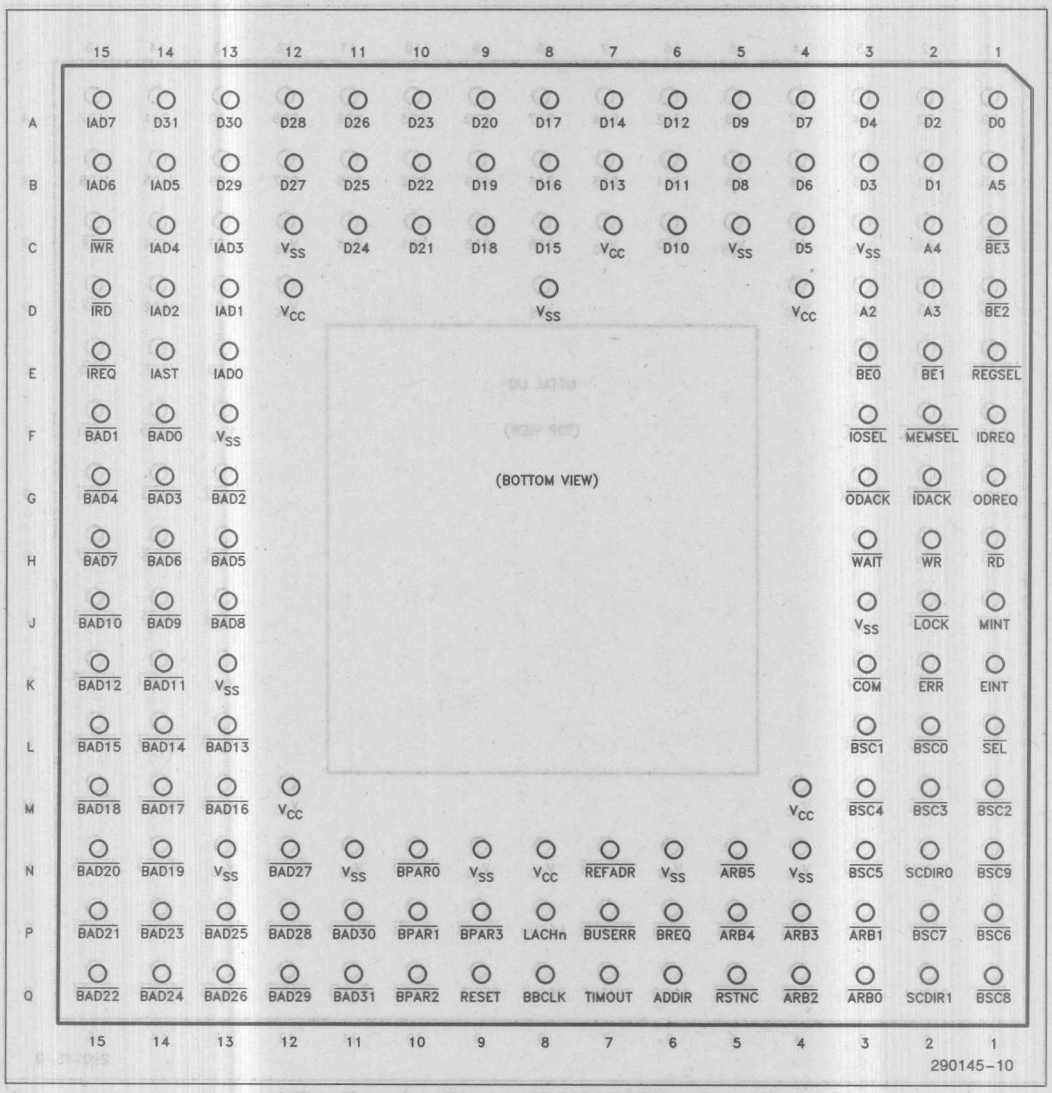


Figure 5-2. MPC 82389 Pinout—View from Pin Side

Table 5-1

Signal and Characteristic		Pin #	Signal and Characteristic		Pin #	Signal and Characteristic		Pin #
V_{CC}		D4	REFADR	O	N7	IAST	I	E14
A5	I	B1	ADDIR	O	Q6	$\overline{IRD}$	I	D15
A4	I	C2	$\overline{BPAR3}$	I/O	P9	$\overline{IWR}$	I	C15
A3	I	D2	$\overline{BAD31}$	I/O	Q11	IAD7	I/O	A15
A2	I	D3	$\overline{BAD30}$	I/O	P11	IAD6	I/O	B15
$\overline{BE3}$	I	C1	$\overline{BAD29}$	I/O	Q12	IAD5	I/O	B14
$\overline{BE2}$	I	D1	$\overline{BAD28}$	I/O	P12	IAD4	I/O	C15
$\overline{BE1}$	I	E2	$\overline{BAD27}$	I/O	N12	IAD3	I/O	C13
$\overline{BE0}$	I	E3	$\overline{BAD26}$	I/O	Q13	IAD2	I/O	D14
$\overline{IOSEL}$	I	F3	$\overline{BAD25}$	I/O	P13	IAD1	I/O	D13
$\overline{MEMSEL}$	I	F2	$\overline{BAD24}$	I/O	Q14	IAD0	I/O	E13
$\overline{REGSEL}$	I	E1	$\overline{BAD23}$	I/O	P14	V_{CC}		D12
$\overline{IDACK}$	I	G2	$\overline{BAD22}$	I/O	Q15	V_{SS}		C12
$\overline{ODACK}$	I	G3	$\overline{BAD21}$	I/O	P15	D31	I/O	A14
IDREQ	O	F1	$\overline{BAD20}$	I/O	N15	D30	I/O	A13
ODREQ	O	G1	$\overline{BAD19}$	I/O	N14	D29	I/O	B13
$\overline{WR}$	I	H2	$\overline{BAD18}$	I/O	M15	D28	I/O	A12
$\overline{RD}$	I	H1	$\overline{BAD17}$	I/O	M14	D27	I/O	B12
$\overline{WAIT}$	O	H3	$\overline{BAD16}$	I/O	M13	D26	I/O	A11
V_{SS}		J3	$\overline{BAD15}$	I/O	Q14	D25	I/O	B11
MINT	O	J1	$\overline{BAD14}$	I/O	L14	D24	I/O	C11
EINT	O	K1	$\overline{BAD13}$	I/O	L13	D23	I/O	A10
$\overline{LOCK}$	I	J2	$\overline{BAD12}$	I/O	K15	D22	I/O	B10
$\overline{ERR}$	O	K2	$\overline{BAD11}$	I/O	K14	D21	I/O	C10
$\overline{SEL}$	O	L1	$\overline{BAD10}$	I/O	J15	D20	I/O	A9
$\overline{COM}$	I	K3	$\overline{BAD9}$	I/O	J14	D19	I/O	B9
$\overline{BSC9}$	I/O	N1	$\overline{BAD8}$	I/O	J13	D18	I/O	C9
$\overline{BSC8}$	I/O	Q1	$\overline{BAD7}$	I/O	H15	D17	I/O	A8
$\overline{BSC7}$	I/O	P2	$\overline{BAD6}$	I/O	H14	D16	I/O	B8
$\overline{BSC6}$	I/O	P1	$\overline{BAD5}$	I/O	H13	D15	I/O	C8
$\overline{BSC5}$	I/O	N3	$\overline{BAD4}$	I/O	G15	D14	I/O	A7
$\overline{BSC4}$	I/O	M3	$\overline{BAD3}$	I/O	G14	D13	I/O	B7
$\overline{BSC3}$	I/O	M2	$\overline{BAD2}$	I/O	G13	D12	I/O	A6
$\overline{BSC2}$	I/O	M1	$\overline{BAD1}$	I/O	F15	D11	I/O	B6
$\overline{BSC1}$	I/O	L3	$\overline{BAD0}$	I/O	F14	D10	I/O	C6
$\overline{BSC0}$	I/O	L2	$\overline{BPAR2}$	I/O	Q10	D9	I/O	A5
SCDIR1	O	Q2	$\overline{BPAR1}$	I/O	P10	D8	I/O	B5
SCDIR0	O	N2	$\overline{BPAR0}$	I/O	N10	D7	I/O	A5
V_{CC}		M4	V_{CC}		N8	D6	I/O	B4
V_{SS}		N4	V_{SS}		N9	D5	I/O	C4
$\overline{ARB5}$	I/O, OC	A15	V_{SS}		N11	D4	I/O	A3
$\overline{ARB4}$	I/O, OC	P5	V_{CC}		M12	D3	I/O	B3
$\overline{ARB3}$	I/O, OC	P4	V_{SS}		N13	D2	I/O	A2
$\overline{ARB2}$	I/O, OC	Q4	V_{SS}		F13	D1	I/O	B2
$\overline{ARB1}$	I/O, OC	P3	V_{SS}		K13	D0	I/O	A1
$\overline{ARB0}$	I/O, OC	Q3	$\overline{BBCLK}$	I	Q8	V_{CC}		C7
V_{SS}		N6	LACHn	I	P8	V_{SS}		D8
$\overline{BREQ}$	I/O, OC	P6	RESET	I	Q9	V_{SS}		C5
TIMOUT	O	Q7	$\overline{RSTNC}$	I/O, OC	Q5	$\overline{BUSERR}$	I/O, OC	P7
$\overline{IREQ}$	O	E15	V_{SS}		C3			

NOTES:
I = Input signal
O = Output signal
I/O = Input or output signal
OC = Open-collector signal

6.0 MPC ELECTRICAL DATA

FOR DC/AC SPECIFICATIONS, PLEASE CONSULT THE LATEST REVISION OF THE DATA SHEET. CALL YOUR NEAREST INTEL DISTRIBUTOR OR THE INTEL SALES OFFICE.

7.0 REFERENCE DOCUMENTS

Part Number	Title Description
176526	MPC User's Manual
146077	MULTIBUS® II Architecture Specifications
149299	Interconnect Interface Specifications
149300	MULTIBUS® II MPC External Product Specifications
149247	MULTIBUS® II Transport Protocol Specifications

intel

APPLICATION NOTE

AP-153

November 1986

Designing with the 8256

CHARLES T. YAGER
APPLICATIONS ENGINEER
INTEL CORPORATION

Order Number: 210907-001

intel

APPLICATION
NOTE

AP-153

November 1986

Designing with the 8256

CHARLES T. YAGER
APPLICATIONS ENGINEER
INTEL CORPORATION

Order Number 210907-001

INTRODUCTION

The INTEL 8256 MUART is a Multifunction Universal Asynchronous Receiver Transmitter designed to be used for serial asynchronous communication while also providing hardware support for parallel I/O, timing, counting and interrupt control. Its versatile design allows it to be directly connected to the MCS®-85, iAPX-86, iAPX-88, iAPX-186, and iAPX-188 microcomputer systems plus the MCS-48 and MCS-51 family of single-chip microcomputers.

The four commonly used peripheral functions contained in the MUART are:

1) Full-duplex, double-buffered serial asynchronous Receiver/Transmitter with an on-chip Baud Rate Generator
2) Two—8-bit parallel I/O ports
3) Five—8-bit counters/timers
4) 8-level priority interrupt controller

This manual can be divided into two parts. The first part describes the MUART in detail, including its functions, registers and pins. This section also describes the interface between the MUART and Intel CPUs plus a discussion on programming considerations. The second section provides an application example: a MUART-based line printer multiplexer. The Appendix contains software listings for the line printer multiplexer and some useful reference information.

DESCRIPTION OF THE MUART

The MUART can be logically partitioned into seven sections: the microprocessor bus interface, the command and status registers, clocking circuitry, asynchronous serial communication, parallel I/O, timer/event counters, and the interrupt controller. This can be seen from the block diagram of the 8256 MUART as shown in Figure 1. The MUART's pin configuration can be seen in Figure 2.

Microprocessor Bus Interface

The microprocessor bus interface is the hardware section of the MUART which allows a μP to communicate with the MUART. It consists of tristate bi-directional data-bus buffers, an address latch, a chip select $\overline{CS}$ latch and bus control logic. In order to provide all of the MUART's functions in a 40-pin DIP while retaining direct register addressing, a multiplexed address/data bus is used.

ADDRESS/DATA BUS

The MUART contains 16 internal directly addressable read/write registers. Four of the eight address/data lines are used to generate the address. When using 8-bit microprocessors such as MCS-85, MCS-48 and MCS-51, AD0–AD3 are used to address the 16 internal registers while Address/Data line 4 (AD4) is not used for addressing. For 16-bit systems, AD1–AD4 are used to generate the address for the internal data registers and AD0 is used as a second active low chip select.

$\overline{RD}$, $\overline{WR}$, $\overline{CS}$

The 8256 bus interface uses the standard bus control signals which are compatible with all Intel peripherals and microprocessors. The chip select signal ($\overline{CS}$), typically derived from an address decoder, is latched along with the address on the falling edge of ALE. As a result, chip select does not have to remain low for the entire bus cycle. However, the data bus buffers will remain tristated unless an $\overline{RD}$ or a $\overline{WR}$ signal becomes active while chip select has been latched in low.

INT, $\overline{INTA}$

The INT and $\overline{INTA}$ signals are used to interrupt the CPU and receive the CPU's acknowledgment to the interrupt request. The MUART can vector the CPU to the appropriate service routine depending on the source of the interrupt.

RESET

When a high level occurs on the RESET pin, the MUART is placed in a known initial state. This initial state is described under "Hardware Reset".

Command and Status Register

There are three command registers and one status register as shown in Figure 1. The three command registers are read/write registers while the status register is a read only. The command registers configure the MUART for its operating environment (i.e., 8 or 16 bits CPU, system clock frequency). In addition, they direct its higher level functions such as controlling the UART, selecting modes of operation for the interrupt controller, and choosing the fundamental frequency for the timers. Command Register 3 is the only register in the MUART which is a bit set/reset register, allowing the programmer to simply perform one write to set or reset any of the bits.

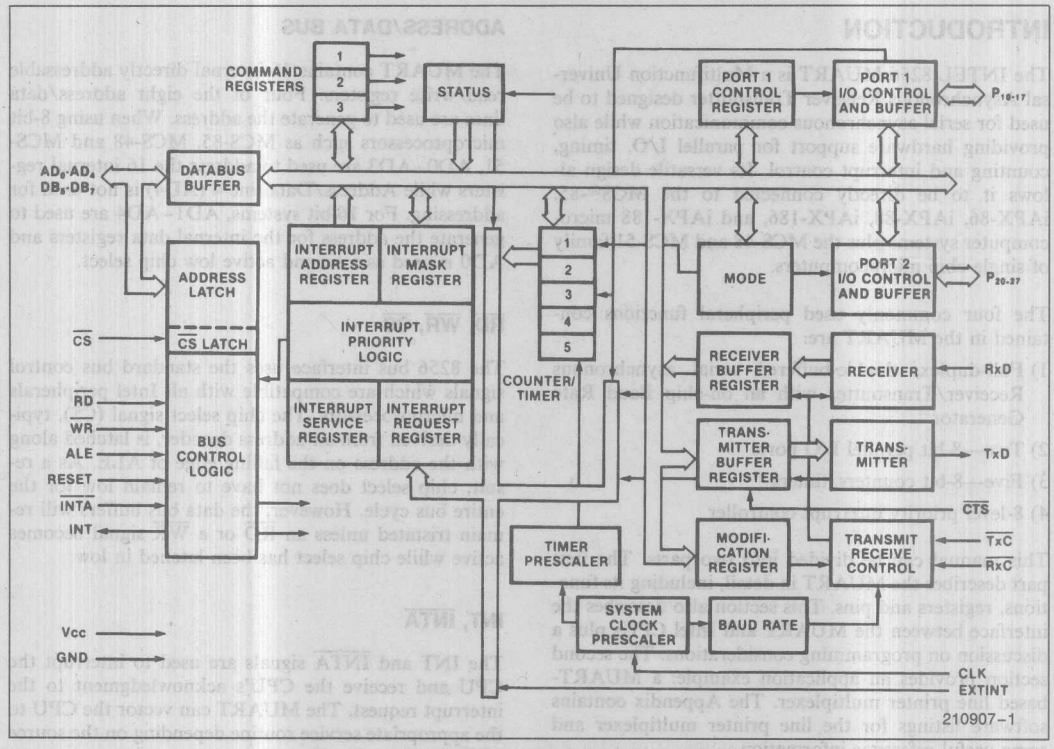


Figure 1. Block Diagram of the 8256 MUART

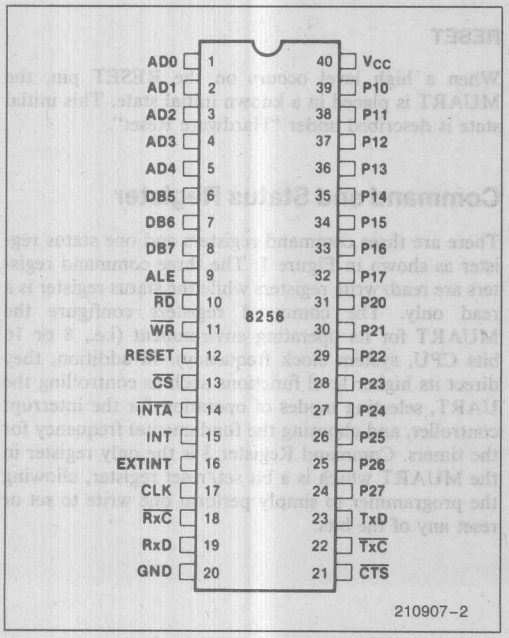


Figure 2. MUART Pin Configuration

The status register provides all of the information about the status of the UART's transmitter and receiver as well as the status of the interrupt pin. The status register is the only read only register in the MUART.

CLOCK CIRCUITRY

The clock for the five timers and baud rate generator is derived from the system clock. The system clock, pin 17 (CLK), is fed into a system clock prescaler which in turn feeds the five timers and the baud rate generator. The MUART's system clock can be asynchronous to the microprocessor's clock.

SYSTEM CLOCK PRESCALER

The system clock prescaler is a programmable divider which normalizes the internal clocking frequency for the timers and baud rate generator to 1.024 MHz. It divides the system clock (CLK) by 1, 2, 3, or 5, allowing clock frequencies of 1.024 MHz, 2.048 MHz, 3.072 MHz or 5.12 MHz. (The commonly used 6.144 MHz crystal frequency for the 8085 results in a 3.072 MHz frequency from the 8085's CLK pin.) If the system clock is not one of the four frequencies mentioned above, then the frequency of the baud rate generator

and the timers will be nonstandard; however, the MUART will still run as long as the system clock meets the data sheet tcy spec.

Timer Prescaler

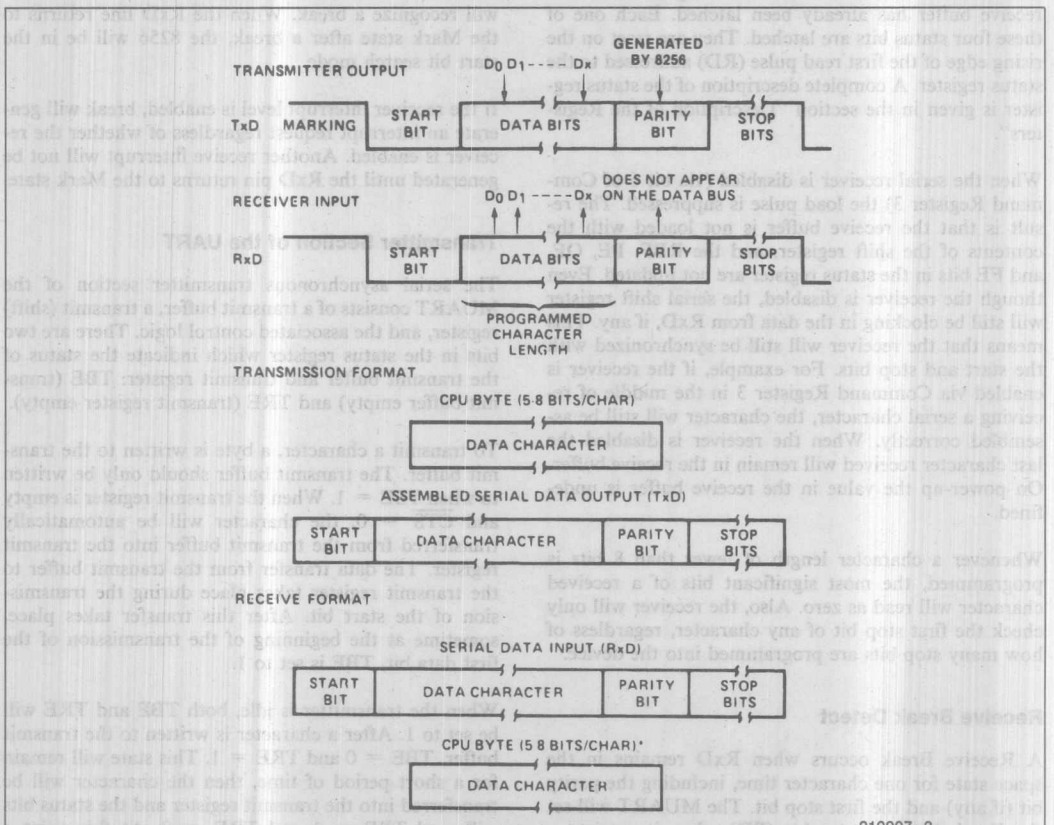
The timer prescaler permits the user to select one of two fundamental timing frequencies for all of the MUART's timers, either 1 kHz or 16 kHz. The frequency selection is made via Command Register 0.

Asynchronous Serial Interface

The asynchronous serial interface of the MUART is a full-duplex double-buffered transmitter and receiver with separate control registers. The standard asynchronous format is used as shown in Figure 3. The operation of the UART section of the MUART is very similar to the operation of the 8251A USART.

Receiver Section of the UART

The serial asynchronous receiver section contains a serial shift register, a receiver buffer register and receiver control logic. The serial input data is clocked into the receive shift register from the RxD pin at the specified baud rate. The sampling actually takes place at the rising edge of RxC, assuming an external clock, or at the rising edge of the internal baud clock. When the receiver is enabled but inactive, the receive logic is sampling RxD at either 32 or 64 times the bit rate, looking for a change from the Mark (high) to the Space (low) state. This is commonly referred to as the start bit search mode. When this state change occurs, the receive logic waits one half of a bit time and then samples RxD again. If RxD is still in the Space state, the receive logic begins to clock in the receive data beginning one bit period later. If RxD has returned to the Mark state (i.e., false start bit), the receive logic will return to the start bit search mode.



*NOTE:
If character length is defined as 5, 6, or 7 bits the unused bits are set to "zero".

Figure 3. Asynchronous Format

Normally the received data is sampled in the center of each bit, however it is possible to adjust the location where the bit is sampled. This feature is controlled by the modification register.

The bit rate of the serial receive data is derived from either the internal baud rate generator or an external clock. When using an external clock, the programmer has a choice of three sampling rates: 1x, 32x, or 64x. Using the internal baud rate generator, the sampling rates are all 64x except for 19.2 Kbps which is 32x.

When the serial shift register clocks in the stop bit, an internal load pulse is generated which transfers the contents of the shift register into the receive buffer. This transfer takes place during the first half of the first stop bit. The load pulse also triggers several other signals relevant to the receive section including Receive Buffer Full (RBF), Parity Error (PE), Overrun Error (OE), and Framing Error (FE). These four status bits are updated after the middle of the first stop bit when the receive buffer has already been latched. Each one of these four status bits are latched. They are reset on the rising edge of the first read pulse (RD) addressed to the status register. A complete description of the status register is given in the section "Description of the Registers".

When the serial receiver is disabled (via bit 6 of Command Register 3) the load pulse is suppressed. The result is that the receive buffer is not loaded with the contents of the shift register, and the RBF, PE, OE, and FE bits in the status register are not updated. Even though the receiver is disabled, the serial shift register will still be clocking in the data from RxD, if any. This means that the receiver will still be synchronized with the start and stop bits. For example, if the receiver is enabled via Command Register 3 in the middle of receiving a serial character, the character will still be assembled correctly. When the receiver is disabled the last character received will remain in the receive buffer. On power-up the value in the receive buffer is undefined.

Whenever a character length of fewer than 8 bits is programmed, the most significant bits of a received character will read as zero. Also, the receiver will only check the first stop bit of any character, regardless of how many stop bits are programmed into the device.

Receive Break Detect

A Receive Break occurs when RxD remains in the space state for one character time, including the parity bit (if any) and the first stop bit. The MUART will set the Break Detect status bit (BD) when it receives a break. The Break Detect status bit is set after the middle of the first stop bit. If the MUART detects a break it will inhibit the receive buffer load pulse, thus the receive buffer will not be loaded with the null character, and none of the four status bits (PE, OE, FE, and RBF) will be updated. The last character received will remain in the receive buffer. A break detect state has the same effect as disabling the receiver—they both inhibit the load pulse—therefore one can think of the break status as disabling the receiver.

The Break Detect status bit is latched. It is cleared by the rising edge of the read pulse addressed to the status register. If a break occurs, and then the RxD data line returns to the Mark state before the status register is read, the BD status bit will remain set until it is read. If RxD returns to the Mark state after the BD status bit has been read true, the BD status bit will be reset automatically without reading the status register.

The receive break detect logic of the MUART is independent of whether the receiver is enabled or disabled; therefore even if the receiver is disabled the MUART will recognize a break. When the RxD line returns to the Mark state after a break, the 8256 will be in the start bit search mode.

If the receiver interrupt level is enabled, break will generate an interrupt request regardless of whether the receiver is enabled. Another receive interrupt will not be generated until the RxD pin returns to the Mark state.

Transmitter Section of the UART

The serial asynchronous transmitter section of the MUART consists of a transmit buffer, a transmit (shift) register, and the associated control logic. There are two bits in the status register which indicate the status of the transmit buffer and transmit register: TBE (transmit buffer empty) and TRE (transmit register empty).

To transmit a character, a byte is written to the transmit buffer. The transmit buffer should only be written to when TBE = 1. When the transmit register is empty and $\overline{CTS}$ = 0, the character will be automatically transferred from the transmit buffer into the transmit register. The data transfer from the transmit buffer to the transmit register takes place during the transmission of the start bit. After this transfer takes place, sometime at the beginning of the transmission of the first data bit, TBE is set to 1.

When the transmitter is idle, both TBE and TRE will be set to 1. After a character is written to the transmit buffer, TBE = 0 and TRE = 1. This state will remain for a short period of time, then the character will be transferred into the transmit register and the status bits will read TBE = 1 and TRE = 0. At this point a second character may be written to the transmit buffer

after which TBE = 0 and TRE = 0. TBE will not be set to 1 again until the transmit register becomes empty and is reloaded with the byte in the transmit buffer.

The transmitter can be disabled only one way—using the $\overline{CTS}$ pin. When $\overline{CTS}$ = 0 the transmitter is enabled, and when $\overline{CTS}$ = 1 the transmitter is disabled. If the transmitter is idle and $\overline{CTS}$ goes from 0 to 1, disabling the transmitter, TBE and TRE will remain set to 1. Since TBE = 1, a character can be written into the transmit buffer. The character will be stored in the transmit buffer but it will not be transferred to the transmit register until $\overline{CTS}$ goes low.

If $\overline{CTS}$ goes from low to high during transmission of a character, the character in transmission will be completed and TxD will return to the Mark state. If the transmitter is full (i.e., TBE and TRE = 0), the transmit shift register will be emptied but the transmit buffer will not; therefore TBE = 0 and TRE = 1.

Transmitter Break Features

The MUART has three transmit break features: Break-In Detect, Transmit Break (TBRK), and Single Character Break (SBRK).

Break-In Detect—A Break-In condition occurs when the MUART is sending a serial message and the transmission line is forced to the space state by the receiving station. Break-In is usually used with half-duplex transmission so that the receiver can signal a break to the transmitter. Port 16 must be connected externally to the transmission line in order to detect a Break-In. If transmission voltage levels other than TTL are used, then proper buffering must be provided so that Port 16 on the MUART will receive the correct polarity and voltage levels.

When Break-In Detect is enabled, Port 16 is polled internally during the transmission of the last or only stop bit of a character. If this pin is low during transmission of the stop bit, the Break Detect status bit (BD) will be set. Break-In Detect and receive Break Detect are OR-ed to set the BD status bit. (Either one can set this bit.) The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on level 5, the transmit interrupt, while Break will generate an interrupt on level 4, the receive interrupt. If RxC and TxC are used for the serial bit rates, Break-In cannot be detected.

Transmit Break—This causes the TxD pin to be forced low for as long as the TBRK bit in Command Register 3 is set. While Transmit Buffer is active, data transfers from the Transmit Buffer to the Transmit register will be inhibited.

If both the Transmit Buffer and the Transmit Register are full, and a Transmit Break command is issued (command register 3, TBRK = 1), the entire character in the Transmit register is sent including the stop bits. TxD is then driven low and the character in the Transmit Buffer remains there until Transmit Break is disabled (command register 3, TBRK = 0). At this time TxD will go high for one bit time and then send the character in the Transmit Buffer.

Single Character Break—This causes TxD to be set low for one character including start bit, data bits, parity bit, and stop bits. The user can send a specific number of Break characters using this feature.

If both the Transmit Buffer and the Transmit Register are full and a Send Break command is issued (command register 3, SBRK = 1) the entire character in the Transmit Register is sent including the stop bits. TxD is driven low for one complete character time followed by a high for two bit times after which the character in the Transmit Buffer is sent.

Modification Register

The modification register is used to alter two standard functions of the receiver (start bit check, and sampling time) and to enable a special indicator flag for half-duplex operation (transmitter status). Disabling start bit check means that the receiver will not return to the start bit search mode if RxD has returned to the Mark state in the center of the start bit. It will simply proceed to assemble a character from the RxD pin regardless of whether it received a false start bit or not. The modification register also allows the user to define where within the receive data bits the MUART will sample.

Parallel I/O

The MUART contains 16 parallel I/O pins which are divided into two 8-bit ports. These two parallel I/O ports (Port 1 and Port 2) can be used for basic digital I/O such as setting a bit high or low, or for byte transfers using a two-wire handshake. Port 1 is bit programmable for input or output, so any combination of the eight bits in Port 1 can be selected as either an input or an output. Port 2 is nibble programmable, which means that all four bits in the upper or lower nibble have to be selected as either inputs or outputs. For byte transfers using the two-wire handshake, Port 2 can either input or output the byte while two bits in Port 1 are used for the handshaking signals.

All of the bits in Port 1 have alternate functions other than I/O ports. As mentioned above, when using the byte handshake mode, two bits on Port 1 are used for the handshaking signals. As a result, these two bits cannot be used for general purpose I/O. The other six bits in Port 1 also have alternate functions if they are not used as I/O ports. Table 1 lists each bit from Port 1 and its corresponding alternate function.

Table 1. Port 1 Control Signals

Pin Symbol	Pin Number	Control Function	Condition
P10 P11	39 38	$\overline{ACK}$ Control Signals for Port 2 $\overline{OBF}$ 8-bit Handshake Output	Mode Register P2C2 – P2C0 = 101
P10 P11	39 38	$\overline{STB}$ Control Signals for Port 2 $\overline{IBF}$ 8-bit Handshake Input	Mode register P2C2 – P2C0 = 100
P12	37	Event Counter 2 Clock Input	Mode Register CT2 = 1
P13	36	Event Counter 3 Clock Input	Mode Register CT3 = 1
P14	35	Internal Baud Rate Generator Clock Output	Mode Word P2C0 – P2C2 = 111 Port 1 Control Word P14 = 1 Command Register 2 B3 – B0 ≥ 3H
P15	34	Timer 5 Trigger Input	Mode Register T5C = 1
P16	33	Break-In Detection Input	Command Register 1 BRKI = 1
P17	32	External Edge Sensitive Interrupt Input	Command Register 1 BITI = 1

The bits in the Port 1 Control Register select whether the pins on Port 1 are inputs or outputs. The pins on Port 1 are selected as control pins through the other programming registers which are relevant to the control signal. Configuring a bit in Port 1 as a control function overrides its definition in the Port 1 Control Register. If the pins on Port 1 are redefined as control signals, the definition of whether the pin is an input or an output in the Port 1 Control Register remains unchanged. If the pins on Port 1 are converted back to I/O pins, they assume the state which was defined in the Port 1 Control Register.

Each parallel I/O port has a latch and drivers. When the port is in the output mode, the data written to the port is latched and driven on the pins. The data which is latched in the I/O ports remains unchanged unless the port is written to again. Reading the ports, whether the port is an input or output, gates the state at the pins onto the data bus. Writing to an input port has no effect on the pin, but the data is stored in the latch and will be output if the direction on the pin is changed later. Writing to a control pin on Port 1 has the same effect as writing to an input pin. If pins 2, 3, 5, and 6 in Port 1 are used for control signals, the contents of the respective output latches will be read, not the state of the control signals. If pins 0, 1, and 7 on Port 1 are used for control signals, the state of the control signals will be read. If pin 4 on Port 1 is used as a test output for the internal baud rate, this clock signal will be output through the output latch, thus the information in the output latch will be lost.

The Two-Wire Byte Handshake

The 8256 can be programmed, via the Mode Register, to implement an input or output two-wire byte handshake. When the Mode Register is programmed for the byte handshake, Port 2 is used to transmit or receive the byte, and pins P10 and P11 are used for the two handshake control signals. Figures 4 and 5 show a block diagram and timing signals for the two-wire handshake input and output.

To set up the two-wire handshake output using interrupts one must first program the Mode Register, and then enable the interrupt via the interrupt mask register. An interrupt will not occur immediately after the two-wire handshake interrupt is enabled. The interrupt is triggered by the rising edge of $\overline{ACK}$. There are two ways to generate the first interrupt. Either the first data byte must be written to Port 2 and completely transferred before an interrupt will occur, or the two-wire handshake interrupt is enabled while $\overline{ACK}$ is low, and then $\overline{ACK}$ goes high.

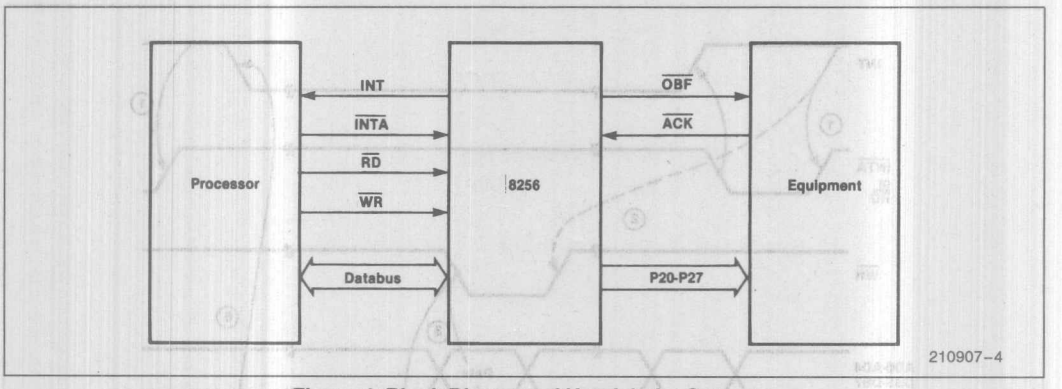


Figure 4. Block Diagram of Handshake Output

Event Counters/Timers

The MUART's five 8-bit programmable counters/timers are binary presettable down counters. The distinction between timer and counter is determined by the clock source. A timer measures an absolute time interval, and its input clock frequency is derived from the MUART's system clock. A counter's input clock frequency is derived from a pulse applied to an external pin. The counter is decremented on the rising edge of this pulse.

When the counters/timers are configured as timers their clock source passes through two dividers: the system clock prescaler, and the timer prescaler. As mentioned before, the system clock prescaler normalizes the internal system clock to 1.024 MHz. The timer prescaler receives this normalized system clock and divides it down to either 1 kHz or 16 kHz, depending on how Command Register 1 is programmed. If more timing resolution is needed the clock frequency can be input externally through the I/O ports.

By programming the Mode Register, four of the 8-bit counters/timers can be cascaded to form two 16-bit counters. Counters/timers 3 and 5 can be cascaded together, and counters/timers 2 and 4 can be cascaded together. Counters/timers 2 and 3 are the lower bytes, while counters/timers 4 and 5 are the upper bytes in the cascaded mode.

Each counter can be loaded with an arbitrary initial value. Timer 5 is the only timer which has a special save register which holds its initial value. Whenever Timer 5 is loaded with an initial value the special save register is also loaded with this value. Timer 5 can be reloaded to its initial value from the detection of a high-to-low transition on Port P15.

The counters are decremented on the first rising edge of the clock after the initial value has been loaded. The setup time for loading the counter when using an external clock is specified in the data sheet. When using internal clocks, the user has no way of knowing the phase relationship of the clock to the write pulse; therefore the timing accuracy is one clock period.

The timers are counting continuously, and an interrupt request is issued any time a single counter or pair of cascaded counters reaches zero. If the timers are going to be used with interrupts, then the programmer should first load the timer with the initial value, then enable the interrupt. If the programmer enables the interrupt first, it is possible that the interrupt will occur before the initial value is loaded. When an interrupt from any one of the timers occurs, the corresponding bit in the interrupt mask register is automatically reset, preventing further interrupt requests from occurring.

The event counters/timers can be used in the following modes of operation:

Timer 1

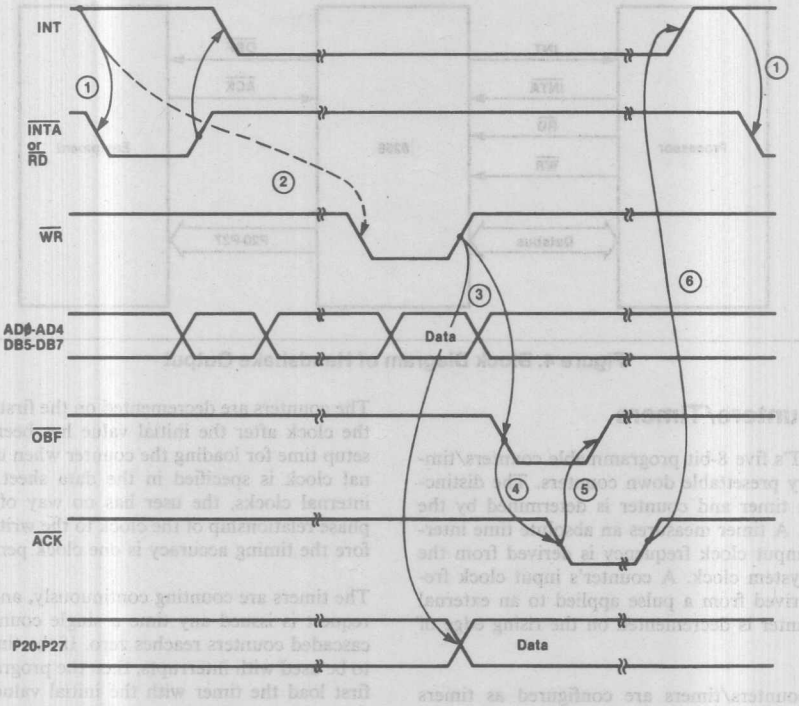
— Serves as an 8-bit timer.

Event Counter/Timer 2

— Serves as an 8-bit timer or event counter, or cascaded with Timer 4 as a 16-bit timer or event counter.

Event Counter/Timer 3

— Serves as an 8-bit timer or event counter, or cascaded with Timer 5 as a 16-bit timer or event counter, with the additional modes of operation selectable for Timer 5.



NOTES:
1. The 8256 signals with INT that the equipment has accepted the last character and that the output latches are empty again.
2. Thereupon, the microprocessor transfers the next data to the 8256.
3. The rising edge of WR latches the data into port 2 (P20 . . . P27) and "Output Buffer Full" (OBF) is set which indicates that a new byte is available.
4. The equipment acknowledges with the falling edge of $\overline{ACK}$ that it recognized $\overline{OBF}$.
5. Thereupon, the 8256 releases $\overline{OBF}$.
6. The equipment acknowledges the data transfer with a rising edge of $\overline{ACK}$ which causes the 8256 to set INT.

Figure 4a. Timing of Handshake Output

Timer 4

— Serves as an 8-bit timer, or cascaded with Event Counter/Timer 2 as a 16-bit timer or event counter.

Timer 5

1. Non-retriggerable 8-bit timer.
2. Retriggerable 8-bit timer whose initial value is loaded from a save register which starts following the negative transition of an external signal. Subsequent transitions of this signal after the counting has started, reloads the initial value and restarts the counting.
3. Cascaded with Event Counter/Timer 3, non-retriggerable 16-bit timer, which can be loaded with an initial value by two write operations.
4. Cascaded with event counter/timer 3, non-retriggerable 16-bit event counter, which can be loaded with an initial value by two write operations.
5. Cascaded with Event Counter/Timer 3, retriggerable 16-bit timer. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the last significant byte (Event Counter/Timer 3) will be set to 0FFH automatically, Loading, starting, and retriggering operations follow the same pattern as in 2.
6. Cascaded with Event Counter/Timer 3, retriggerable 16-bit event counter. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to 0FFH automatically. Loading, starting, and retriggering operations follow the same pattern as in 2.

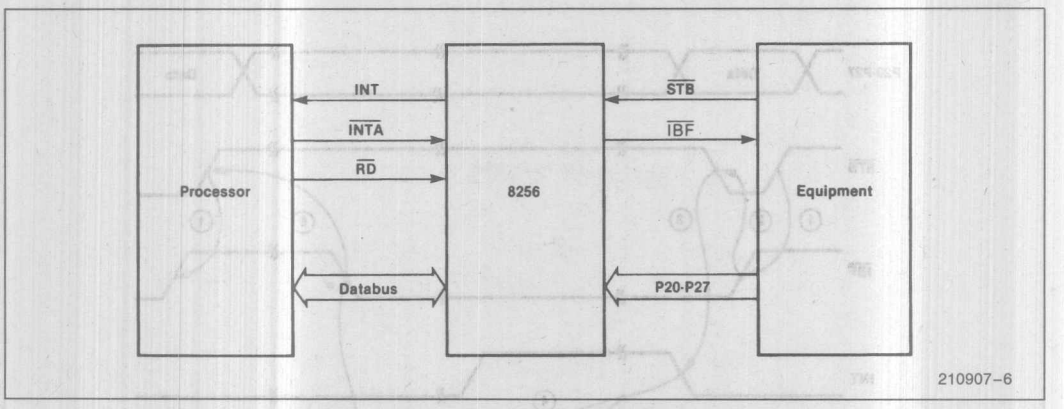


Figure 5. Block Diagram of Handshake Input

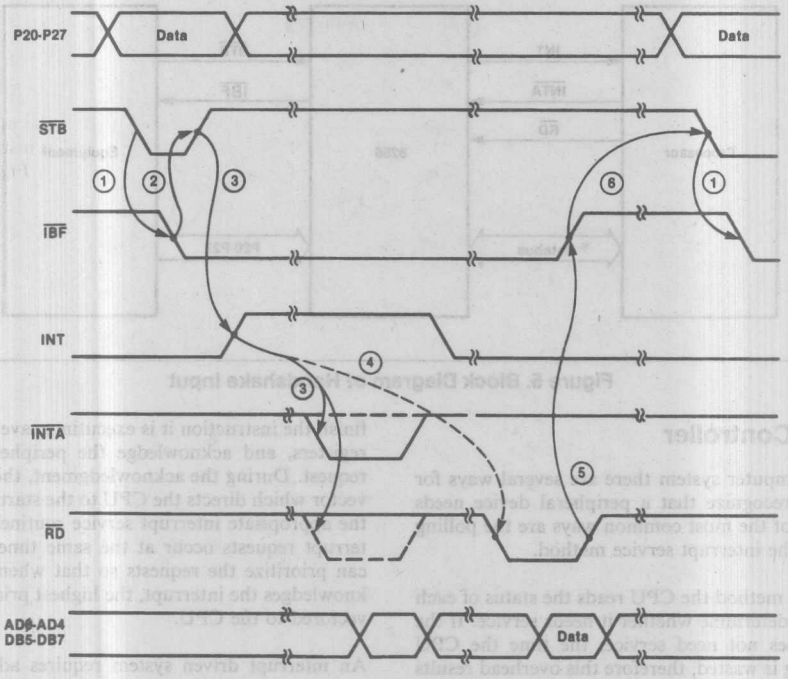
Interrupt Controller

In a microcomputer system there are several ways for the CPU to recognize that a peripheral device needs service. Two of the most common ways are the polling method and the interrupt service method.

In the polling method the CPU reads the status of each peripheral to determine whether it needs service. If the peripheral does not need service, the time the CPU spends polling is wasted; therefore this overhead results in increasing the execution time. Some systems must meet a specific request to response time such as a real time signal. In this case the programmer must guarantee that the peripheral is polled at a certain frequency. This polling frequency cannot always easily be met when the CPU must execute a main program as well as subroutines. Usually each peripheral has its own request to response time requirements; therefore the user must establish a priority scheme.

The interrupt method provides certain advantages over the polling method. When a peripheral device needs service it signals the CPU through hardware asynchronously, thus reducing the overhead of polling a device which does not need service. The CPU would typically finish the instruction it is executing, save the important registers, and acknowledge the peripheral's interrupt request. During the acknowledgment, the CPU reads a vector which directs the CPU to the starting location of the appropriate interrupt service routine. If several interrupt requests occur at the same time, special logic can prioritize the requests so that when the CPU acknowledges the interrupt, the highest priority request is vectored to the CPU.

An interrupt driven system requires additional hardware to control the interrupt request signal, priority, and vectoring. The 8256 integrates this additional hardware onto the chip. The interrupt controller on the MUART is directly compatible with the MCS-85, iAPX-86, iAPX-88, iAPX-186, iAPX-188 family of microcomputer systems, and it can also be used with other microprocessors as well. It contains eight priority levels, however, there are a total of 12 interruptable sources: 10 internal and 2 external. Since there are eight priority levels, only eight interrupts can be used at one time. The assignment of the interrupts used is selected by Command Register 1 and by the mode register. The MUART's interrupt sources have a fixed priority. Table 2 displays how the 12 interrupt sources are mapped into the 8 priority levels.



210907-7

NOTES:

1. The equipment indicates with the falling edge of $\overline{STB}$ (Strobe) that a new character is available at port 2. The 8256 acknowledges the indication by activating $\overline{IBF}$ (Input Buffer Full).
2. Thereupon, the equipment releases $\overline{STB}$ and the 8256 latches the character.
3. The 8256 informs the microprocessor through INT that a new character is ready for transfer.
4. The microprocessor reads the character.
5. The rising edge of signal $\overline{RD}$ resets signal $\overline{IBF}$.
6. The action signals to the equipment that the input latches of the 8256 are empty and the next character can be transferred.

Figure 5a. Timing for Handshake Input

Table 2. Mapping of Interrupt Sources to Priority Levels

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

MCS®-85/8256 Interrupt Operation

The 8256 is compatible with the 8085 interrupt vectoring method when the 8086 bit in Command Register 1 of the MUART is set to 0. This is the default condition after a hardware reset. The 8085 has five hardware interrupt pins: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP. When the MUART's interrupt acknowledge feature is enabled (IAE bit 5 Command Register 3 = 1) the MUART's INT Pin 15 should be tied to the 8085's INTR, and both the 8085 and the MUART's INTA pins should be tied together. All of the interrupt pins on the 8085 except INTR automatically vector the program counter to a specified location in memory.

When the INTR pin becomes active (HIGH), assuming the 8085 has interrupts enabled, the 8085 fetches the next instruction from the data bus where it has been placed by the 8256 or some other interrupt controller. This instruction is usually a Call or an RST0 through RST7. Figure 6 shows the memory locations where the 8085 will vector to based on which type of interrupt occurred.

The 8085 can receive an interrupt request any time, since its INTR input is asynchronous. The 8085, however, doesn't always acknowledge an interrupt request immediately. It can accept or disregard requests under software control using the EI (Enable Interrupt) or DI (Disable Interrupt) instructions.

At the end of each instruction cycle, the 8085 examines the state of its INTR pin. If an interrupt request is present and interrupts are enabled, the 8085 enters an interrupt machine cycle. During the interrupt machine cycle the 8085 automatically disables further interrupts until the EI instruction is executed. Unlike normal machine cycles, the interrupt machine cycle doesn't increment the program counter. This ensures that the 8085 can return to the pre-interrupt program location after the interrupt service is completed. The 8085 issues an $\overline{\text{INTA}}$ pulse indicating that it is honoring the request and is ready to process the interrupt.

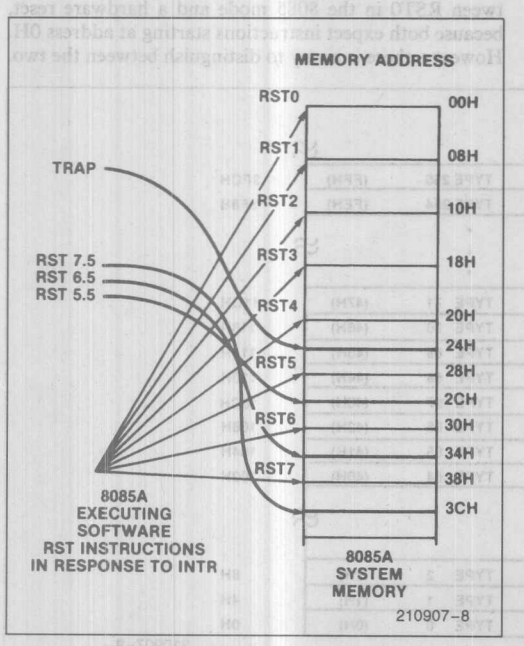


Figure 6. 8085A Hardware and Software RST Branch Locations

The 8256 can now vector program execution to the corresponding service routine. This is done during the first and only $\overline{\text{INTA}}$ pulse. Upon receiving the $\overline{\text{INTA}}$ pulse, the 8256 places the opcode RSTn on the data bus; where n equals 0 through 7 based on the level of the interrupt requested. The RSTn instruction causes the contents of the program counter to be pushed onto the stack, then transfers control to the instruction whose address is eight times n, as shown in Figure 6.

Note that because interrupts are disabled during the interrupt acknowledge sequence, the EI instruction must be executed in either the service routine or the main program before further interrupts can be processed.

For additional information on the 8085 interrupt operation and the RSTn instruction, refer to the *MCS-85 User's Manual.*

iAPX-86/88—8256 Interrupt Operation

The MUART is compatible with the 8086/8088 method of interrupt vectoring when the 8086 bit in Command Register 1 is set to 1. The MUART's $\overline{\text{INT}}$ pin is tied to the 8086/8088 INTR pin, and its $\overline{\text{INTA}}$ pin connected to the 8086/88's $\overline{\text{INTA}}$ pin. Like the 8085, the 8086/8088's INTR pin is also asynchronous so that an interrupt request can occur at any time. The 8086/8088 can accept or disregard requests on the INTR pin under software control instructions. These instructions set or clear the interrupt-enabled flag IF. When the 8086/8088 is powered-on or reset, the IF flag is cleared, disabling external interrupts on INTR.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different from the 8085. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in PUSHF instruction). It then clears the IF flag, which disables interrupts. Finally, the contents of both the code segment register and the instruction pointer are pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and program location which are used to return from the service routine. The 8086/8088 then issues the first of two INTA pulses which signals the 8256 that the 8086/8088 has honored its interrupt request.

The 8256 is now ready to vector program execution to the appropriate service routine. Unlike the 8085 where the first INTA pulse is used to place an instruction on the data bus, the first INTA pulse from the 8086/8088 is used only to signal the 8256 of the honored request. The second INTA pulse causes the 8256 to place a single interrupt vector byte onto the data bus. The 8256 places the interrupt vector bytes 40H through 47H cor-

responding to the level of the interrupt to be serviced. Not used as a direct address, this interrupt vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt vector table. Each type in the interrupt vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 7 shows a block diagram of the interrupt vector table. When the 8086/8088 receives an interrupt vector byte, it multiplies its value by four to acquire the address of the interrupt type.

Once the service routine is completed the main program may be re-entered by using an IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag; thus interrupts are re-enabled automatically when returning from the service routine. For further information refer to the *iAPX 86,88 User's Manual.*

Using the 8256's Interrupt Controller without $\overline{\text{INTA}}$

There are several configurations where the 8256 will not have an $\overline{\text{INTA}}$ signal connected to it. Some examples are when using the 8256 with an 8051 or 8048, or when connecting the INT pin on the 8256 to the 8085's RST 7.5, RST 6.5, or RST 5.5 inputs. In these configurations the IAE bit in Command Register 3 is set to 0, and the $\overline{\text{INTA}}$ pin on the 8256 is tied high. When the interrupt occurs the CPU should branch to a service routine which reads the interrupt address register to determine which interrupt request level occurred. The interrupt address register contains the level of the interrupt multiplied by four. Reading the interrupt address register is equivalent in effect to the $\overline{\text{INTA}}$ signal; it clears the INT pin and indicates to the MUART that the interrupt request has been acknowledged. After the CPU reads the value in the interrupt address register, it can add an offset to this value and branch to an interrupt vector table which contains jump instructions to the appropriate interrupt service routines. An 8085 program which demonstrates this routine is given is Figure 8.

Table 3 summarizes the priority levels and the interrupt vectors which the 8256 sends back to the CPU. Note that when using Timer 1 there is a conflict present between RST0 in the 8085 mode and a hardware reset, because both expect instructions starting at address 0H. However, there is a way to distinguish between the two.

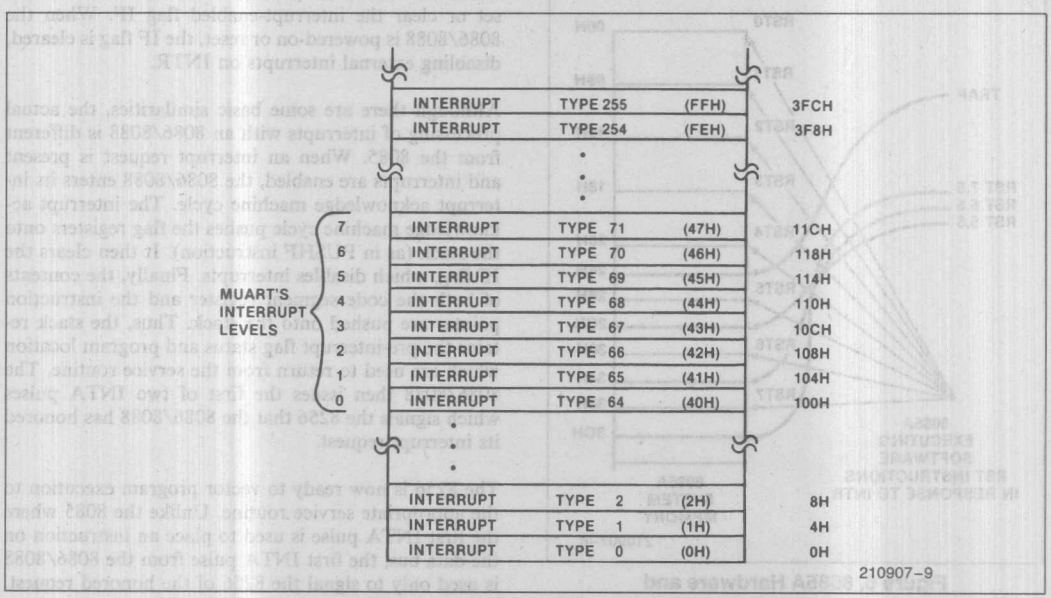


Figure 7. 8086/8088 Interrupt Vector Table

```
INTA: IN   INTADD      ;Read the Interrupt Address Register
      MOV  L, A        ;Put the interrupt address in HL
      XRA  A
      MOV  H, A

      LXI  B, TABLE    ;Load BE with the interrupt table offset
      DAD  B           ;Add the offset to the interrupt address
      PCHL             ;Jump to the interrupt vector table
```

Figure 8. Software Interrupt Acknowledge Routine

Table 3. Assignment of Interrupt Levels to Interrupt Sources

Interrupt Level	Restart Command 8085 Mode	Interrupt Vector 8086 Mode	Interrupt Address	Trigger Mode	Sources (Only One Source Can Be Assigned at Any Time)	Selection By
Highest Priority 0	RST0	40H	0H	Edge	Timer 1	—
1	RST1	41H	4H	Edge	Event Counter/Timer 2 or External Interrupt Request on Port 1 P17	Command Word 1 BITI (Bit 2)
2	RST2	42H	8H	Level	Input EXTINT	—
3	RST3	43H	CH	Edge	Event Counter/Timer 3 or Cascaded Event Counters/ Timers 3 and 5	Mode Word T35 (Bit 7)
4	RST4	44H	10H	Edge	Serial Receiver	—
5	RST5	45H	14H	Edge	Serial Transmitter	—
6	RST6	46H	28H	Edge	Timer 4 or Cascaded Event Counters/Timers 2 and 4	Mode Word T24 (Bit 6)
7 Lowest Priority	RST7	47H	1CH	Edge	Timer 5 or Port 2 with Handshaking Interrupt Request	Mode Word P2C2–P2C0 (Bits 2 . . . 0)

NOTE:
1. If no interrupt requests are pending and $\overline{\text{INTA}}$ cycle occurs, interrupt level 2 will be the default value vectored to the CPU.

After a hardware reset, all control registers are reset to a value of 0H; therefore when using Timer 1, Reset and RST0 can be distinguished by reading one of the control registers of the 8256 which has not been programmed with a value of 0H. The control registers will contain the previously programmed values if RST0 occurs.

Interrupt Registers

The 8256's interrupt controller has several registers associated with it: an Interrupt Mask Register, an Interrupt Address Register, an Interrupt Request Register, an Interrupt Service Register, and a Priority Controller. Only the Interrupt Mask Registers and the Interrupt Address Register can be accessed by the user.

Interrupt Mask Registers

The Interrupt Mask Registers consist of two write registers—the Set Interrupts Register and Reset Interrupts Register, and one read register—the Interrupt Enable Register. Each one of the eight levels of interrupts may be individually enabled or disabled through these registers. Writing a one to any of the bits in the Set Interrupts Register enables the corresponding interrupt level, while writing a one to a bit in the Reset Interrupts Register disables the corresponding interrupt level. Reading the Interrupt Enable Register allows the user to determine which interrupt levels are enabled. The bits which are set to one in the Interrupt Enable Register correspond to the levels which are enabled. All of the interrupt levels will remain enabled until disabled by the Reset Interrupts Register except the counter/timer interrupts which automatically disable themselves when they reach zero.

Interrupt requests occurring when the corresponding interrupt level is disabled are lost. An interrupt will only occur if the interrupt is enabled before the interrupt request occurs.

Interrupt Address Register

The Interrupt Address Register contains an identifier for the currently requested interrupt level. The numerical value in this register is equal to the interrupt level multiplied by four. It can be used in lieu of an $\overline{\text{INTA}}$ signal to vector the CPU to the appropriate interrupt service routine. Reading this register has the same effect as the $\overline{\text{INTA}}$ pulse: it clears the INT pin and indicates an interrupt acknowledgement to the MUART. If the Interrupt Address Register is read while no interrupts are pending, the external interrupt EXTINT will be the default value, 08H.

Interrupt Request Register

The Interrupt Request Register latches all pending interrupt requests unless they are masked off. The request is set whenever the associated event occurs.

Interrupt Service Register

In the fully nested mode of operation, every interrupt request which is granted service is entered into this register. The appropriate bit will be set whenever the interrupt is acknowledged by $\overline{\text{INTA}}$ or by reading the Interrupt Address Register. At the same time, the corresponding bit in the Interrupt Request Register is reset. The Interrupt Service Register bit remains set until the microcomputer transfers the End Of Interrupt command (EOI) to the device by writing it into Command Register 3. In the normal mode the bits in the Interrupt Service Register are never set.

Priority Controller

The priority controller selects the highest priority request in the Interrupt Request Register from up to eight requests pending. If the $\overline{\text{INTA}}$ signal is enabled and becomes active, the priority controller will cause the highest priority level in the Interrupt Request Register to be vectored back to the CPU, regardless of whether the 8256 is in the normal mode or the nested mode. In the normal mode, if any bits are set in the Interrupt Request Register, the INT pin is activated. The highest priority level in the Interrupt Request Register will be transferred to the Interrupt Address Register at the same time the interrupt request occurs. In the Fully Nested mode, the priorities of all pending requests are compared to the priorities in the Interrupt Service Register. If there is a higher priority in the Interrupt Request Register than in the Interrupt Service Register, the INT signal will be activated and the new interrupt level will be loaded into the Interrupt Address Register.

Interrupt Modes

There are two modes of operation for the interrupt controller: a normal mode and a fully nested mode. In the normal mode the CPU should only be a maximum of one interrupt level deep; therefore, the CPU can be interrupted only while in the main program and not while in an interrupt service routine. In the fully nested mode it is possible for the CPU to be nested up to eight interrupt levels deep. Using the fully nested mode, the MUART will activate the INT pin only when a higher priority than the one in service is requested. The fully nested mode is used to protect high priority interrupt service routines from being interrupted by equal or lower priority requests.

Normal Mode

In the normal mode of operation the 8256 will activate the INT pin whenever any of the bits in the Interrupt Request Register are set. The bits in the Interrupt Request Register can be set only if the corresponding interrupts are enabled. If more than one interrupt request bit is set, the MUART will always place the highest priority level in the Interrupt Address Register and vector this level to the CPU during an $\overline{\text{INTA}}$ cycle. When the CPU acknowledges the interrupt request, using either the $\overline{\text{INTA}}$ signal or by reading the Interrupt Address Register, the corresponding Interrupt Request Register bit is reset. Since the Interrupt Service Register bits are never set, there is no indication in the MUART that an interrupt service routine is in progress. Therefore, the priority controller will interrupt the CPU again if any of the interrupt request bits are set, regardless of whether the next request is a higher, lower, or equal priority.

The implied way to design a program using the normal mode is to have the CPU's interrupt flag enabled during portions of the main program, but to leave the interrupt flag disabled while the CPU is executing code in an interrupt service routine. This way, the CPU can never be interrupted in an interrupt service routine. Upon completion of an interrupt service routine the program can enable the CPU's interrupt flag, then return to the main program.

Figure 9 shows an example of how the normal mode of interrupts may operate. As the CPU begins executing code in the main program, certain I/O ports, variables,

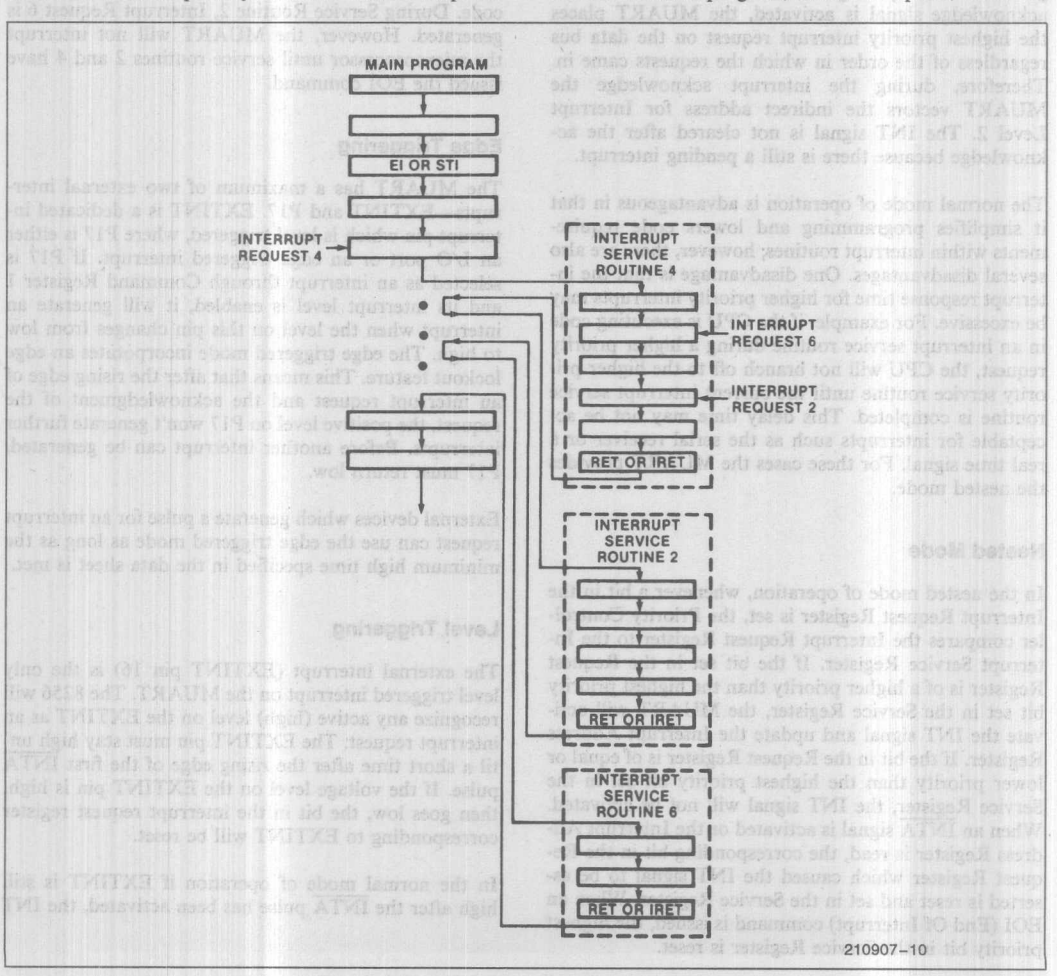


Figure 9. Normal Interrupt Mode Example

and arrays need to be initialized. During this time the CPU's interrupt flag is disabled. Once the program has completed the initialization routine and can accept an interrupt, the interrupt flag is enabled. In the 8085 this is done with the assembly language instruction EI, and on the 8086 with STI.

A short time later, an interrupt request comes in on Level 4. Since the CPU's interrupt flag is enabled, the interrupt acknowledge signal is activated and the CPU branches off to Interrupt Service Routine 4. While the CPU is executing code in Interrupt Service Routine 4, an interrupt request comes in on Level 6 and then a short time later on Level 2. The 8256 activates the INT signal; however, the CPU ignores this because its interrupt flag is disabled. Upon returning to the main program the interrupt flag is enabled. When the interrupt acknowledge signal is activated, the MUART places the highest priority interrupt request on the data bus regardless of the order in which the requests came in. Therefore, during the interrupt acknowledge the MUART vectors the indirect address for Interrupt Level 2. The INT signal is not cleared after the acknowledge because there is still a pending interrupt.

The normal mode of operation is advantageous in that it simplifies programming and lowers code requirements within interrupt routines; however, there are also several disadvantages. One disadvantage is that the interrupt response time for higher priority interrupts may be excessive. For example, if the CPU is executing code in an interrupt service routine during a higher priority request, the CPU will not branch off to the higher priority service routine until the current interrupt service routine is completed. This delay time may not be acceptable for interrupts such as the serial receiver or a real time signal. For these cases the MUART provides the nested mode.

Nested Mode

In the nested mode of operation, whenever a bit in the Interrupt Request Register is set, the Priority Controller compares the Interrupt Request Register to the Interrupt Service Register. If the bit set in the Request Register is of a higher priority than the highest priority bit set in the Service Register, the MUART will activate the INT signal and update the Interrupt Address Register. If the bit in the Request Register is of equal or lower priority than the highest priority bit set in the Service Register, the INT signal will not be activated. When an $\overline{\text{INTA}}$ signal is activated or the Interrupt Address Register is read, the corresponding bit in the Request Register which caused the INT signal to be asserted is reset and set in the Service Register. When an EOI (End Of Interrupt) command is issued, the highest priority bit in the Service Register is reset.

Figure 10 shows an example of the program flow using the nested mode of interrupts. During the main program an interrupt request is generated from Level 4. Since the interrupt flag is enabled, the interrupt acknowledge signal is activated, and the microprocessor is vectored to Service Routine 4. During Service Routine 4, Level 2 requests an interrupt. Since Level 2 is a higher priority than Level 4, the 8256 activates its INT signal. An interrupt acknowledge is not generated because the interrupt flag is disabled. This section of code in Service Routine 4 is protected and cannot be interrupted. A protected section of code may reinitialize a timer, take a sample, or update a global variable. When the interrupt flag is enabled the microprocessor acknowledges the interrupt and vectors into Service Routine 2. Service Routine 2 immediately enables the interrupt flag because it does not have a protected section of code. During Service Routine 2, Interrupt Request 6 is generated. However, the MUART will not interrupt the microprocessor until service routines 2 and 4 have issued the EOI command.

Edge Triggering

The MUART has a maximum of two external interrupts—EXTINT and P17. EXTINT is a dedicated interrupt pin which is level triggered, where P17 is either an I/O port or an edge triggered interrupt. If P17 is selected as an interrupt through Command Register 1 and its interrupt level is enabled, it will generate an interrupt when the level on this pin changes from low to high. The edge triggered mode incorporates an edge lockout feature. This means that after the rising edge of an interrupt request and the acknowledgment of the request, the positive level on P17 won't generate further interrupts. Before another interrupt can be generated, P17 must return low.

External devices which generate a pulse for an interrupt request can use the edge triggered mode as long as the minimum high time specified in the data sheet is met.

Level Triggering

The external interrupt (EXTINT pin 16) is the only level triggered interrupt on the MUART. The 8256 will recognize any active (high) level on the EXTINT as an interrupt request. The EXTINT pin must stay high until a short time after the rising edge of the first $\overline{\text{INTA}}$ pulse. If the voltage level on the EXTINT pin is high, then goes low, the bit in the interrupt request register corresponding to EXTINT will be reset.

In the normal mode of operation if EXTINT is still high after the $\overline{\text{INTA}}$ pulse has been activated, the INT

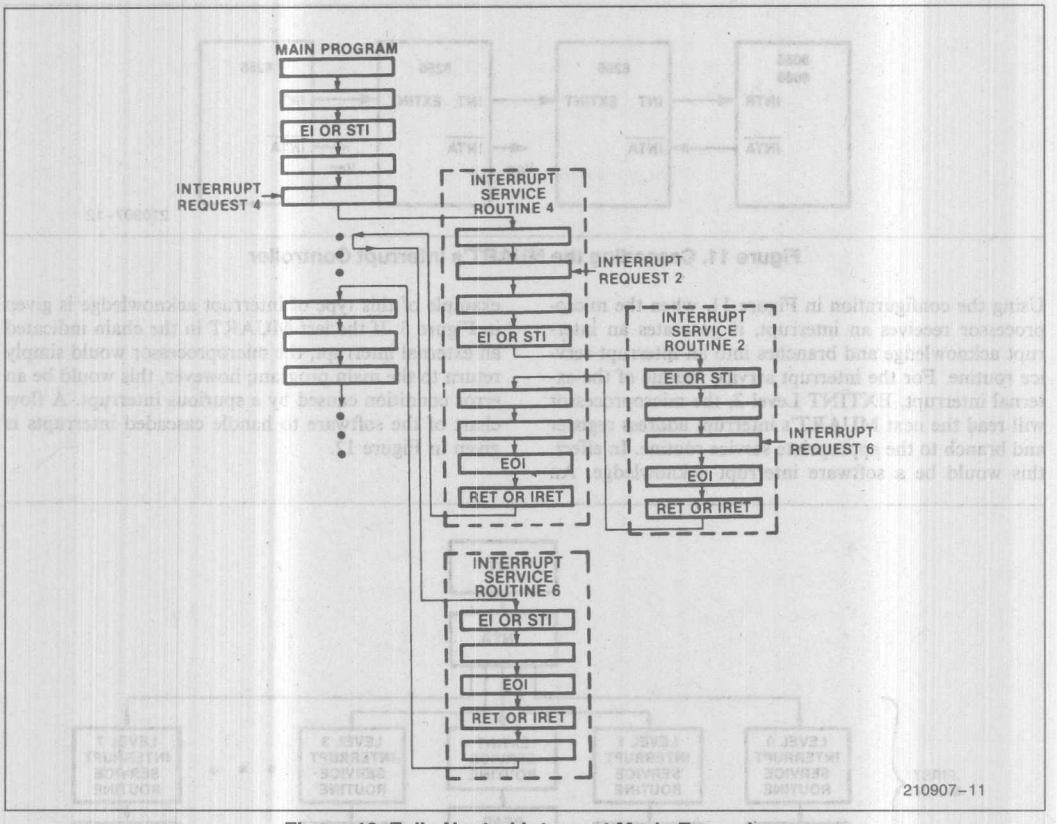


Figure 10. Fully Nested Interrupt Mode Example

signal will remain active. If the microprocessor's interrupt flag is immediately re-enabled, another interrupt will occur. Unless repeated interrupt generation is desired, the programmer should not re-enable the CPU's interrupt flag until EXTINT has gone low.

In the nested mode of operation, if EXTINT is still high after the $\overline{\text{INTA}}$ pulse has been activated, the INT signal will not be reactivated. This is because in the nested mode only a higher priority interrupt than the one being serviced can activate the INT signal. The EXTINT pin should go inactive (low) before the EOI command is issued if an immediate interrupt is not desired.

Depending upon the particuar design and application, the EXTINT pin has a number of uses. For example, it can provide repeated interrupt generation in the normal mode. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another use of the EXTINT pin is that a number of external interrupt requests can be wire-ORed. This can't be done using P17, for if a device makes an interrupt request while P17 is high (from another request), its transition will be shadowed. Note that when a wire-OR'ed scheme is used, the actual requesting device has to be determined by the software in the service routine.

Cascading the MUART's Interrupt Controller

Cascading the MUART's interrupt controller is necessary in an interrupt driven system which contains more than one interrupt controller, such as a system using more than one MUART, or using a MUART with another interrupt controller like the 8259A. For a system which uses several MUART's, one of them is tied directly to the microprocessor's INT and $\overline{\text{INTA}}$ pins, while the remaining MUARTs are daisy-chained using the EXTINT and INT pins. This is shown in Figure 11.

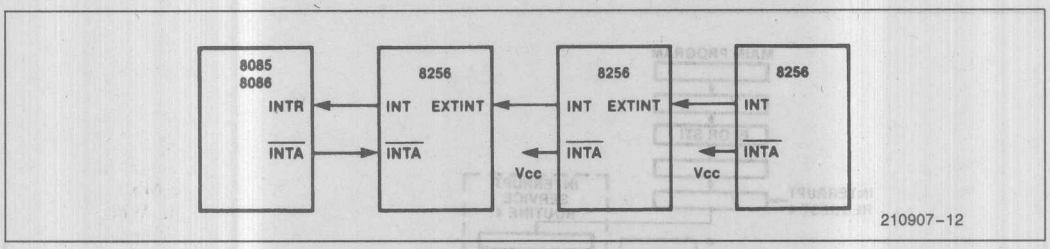


Figure 11. Cascading the MUART's Interrupt Controller

Using the configuration in Figure 11, when the microprocessor receives an interrupt, it generates an interrupt acknowledge and branches into an interrupt service routine. For the interrupt service routine of the external interrupt, EXTINT Level 2, the microprocessor will read the next MUART's interrupt address register and branch to the appropriate service routine. In effect, this would be a software interrupt acknowledge. An example of this type of interrupt acknowledge is given in Figure 8. If the last MUART in the chain indicated an external interrupt, the microprocessor would simply return to the main program; however, this would be an error condition caused by a spurious interrupt. A flow chart of the software to handle cascaded interrupts is given in Figure 12.

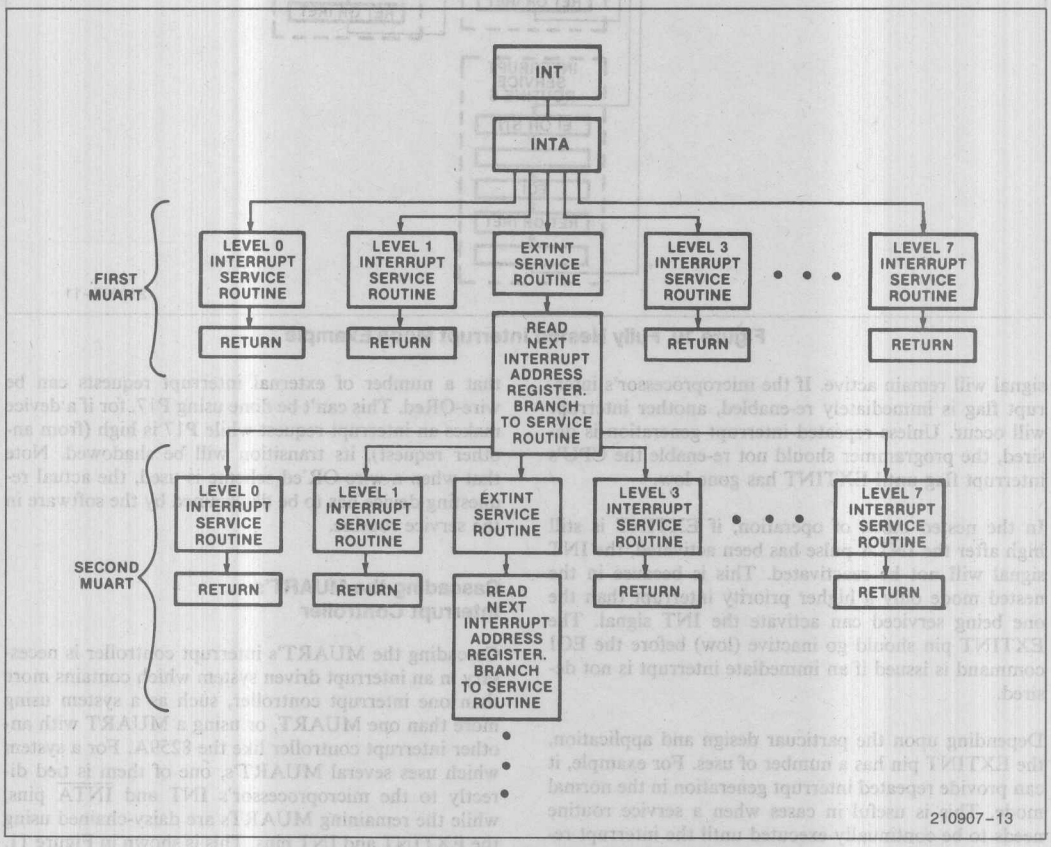


Figure 12. Flow Chart to Resolve Interrupt Request When Cascading MUART Interrupt Controllers

Some consideration should be given to the priority of the interrupts when cascading MUARTs. If all of the MUART's Level 0 and Level 1 interrupts are disabled, the highest priority interrupt is the EXTINT. In this case the last MUART in the chain would have the highest priority; however, it would take the longest time to propagate back to the CPU. If, however, Level 0 or Level 1 interrupts were enabled, the closer to the microprocessor the MUART is, the higher the priority these two levels would have.

When using the 8256 interrupt controller along with some other interrupt controller, such as the 8259A, the MUART's INT signal would simply be tied to one of the interrupt controller's request inputs. The service routine for the MUART's interrupt request would initially perform the software interrupt acknowledge before servicing the MUART's interrupt request. A block diagram of this configuration is given in Figure 13.

Polling the MUART

If interrupts are not used, the only other way to control the MUART is to poll it. It is still possible to use the priority structure of the MUART with polling. In this mode of operation the MUART's INT signal (Pin 15) is not used, and the $\overline{\text{INTA}}$ pin is tied high. Since the INT pin's level is duplicated in the MSB of the Status Register, a program can poll this bit. When it becomes set, the program could read the Interrupt Address Register to determine the cause. Either the normal or nested mode of operation can be used. Note that the functions used with this polled method must have their interrupts enabled.

It is also possible to poll the counters/timers, parallel I/O, and UART separately. To control the UART, one could poll the Status Register. Byte handshakes with the parallel I/O can be controlled by polling Port 1. Finally, each counter/timer has its own register which can be polled.

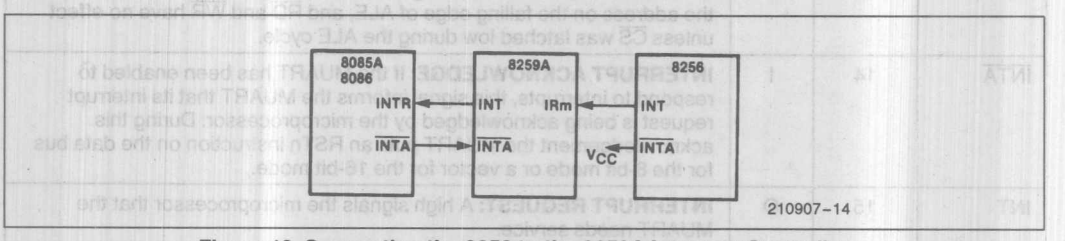


Figure 13. Connecting the 8256 to the 8259A Interrupt Controller

PIN DESCRIPTIONS

Symbol	Pin No.	Type	Name and Function
AD0–AD4 DB5–DB7	1–5 6–8	I/O	**ADDRESS/DATA:** Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, AD0–AD3 are used to select the proper register, while AD1–AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.
ALE	9	I	**ADDRESS LATCH ENABLE:** Latches the 5 address lines on AD0–AD4 and $\overline{CS}$ on the falling edge.
$\overline{RD}$	10	I	**READ CONTROL:** When this signal is low, the selected register is gated onto the data bus.
$\overline{WR}$	11	I	**WRITE CONTROL:** When this signal is low, the value on the data bus is written into the selected register.
RESET	12	I	**RESET:** An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.
$\overline{CS}$	13	I	**CHIP SELECT:** A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and $\overline{RD}$ and $\overline{WR}$ have no effect unless $\overline{CS}$ was latched low during the ALE cycle.
$\overline{INTA}$	14	I	**INTERRUPT ACKNOWLEDGE:** If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.
INT	15	O	**INTERRUPT REQUEST:** A high signals the microprocessor that the MUART needs service.
EXTINT	16	I	**EXTERNAL INTERRUPT:** An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an $\overline{INTA}$ occurs or the interrupt address register is read.
CLK	17	I	**SYSTEM CLOCK:** The reference clock for the baud rate generator and the timers.
RxC	18	I/O	**RECEIVE CLOCK:** If the baud rate bits in Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1–0FH, this pin outputs a square wave whose rising edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.
RxD	19	I	**RECEIVE DATA:** Serial data input.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{CTS}$	21	I	**CLEAR TO SEND:** This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected, $\overline{CTS}$ is level sensitive. As long as $\overline{CTS}$ is low, any character loaded into the transmitter buffer register will be transmitted serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1–0FH is selected, $\overline{CTS}$ must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where 1/2 of the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. IF $\overline{CTS}$ is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on $\overline{CTS}$ occurs. If 0.75 stop bits is chosen, the $\overline{CTS}$ input is edge sensitive. A negative edge on $\overline{CTS}$ results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on $\overline{CTS}$. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.
TxC	22	I/O	**TRANSMIT CLOCK:** If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3–0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1 1/2 or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.
TxD	23	O	**TRANSMIT DATA:** Serial data output.
P27–P20	24–31	I/O	**PARALLEL I/O PORT 2:** Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.
P17–P10	32–39	I/O	**PARALLEL I/O PORT 1:** Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.
GND	20	PS	**GROUND:** Power supply and logic ground reference.
V_{CC}	40	PS	**POWER:** +5V power supply.

DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 4 lists the registers and their addresses.

Command Register 1

L1	L0	S1	S0	BRKI	BITI	8086	FRQ
(0R)				(0W)			

FRQ—Timer Frequency Select

This bit selects between two frequencies for the five timers. If FRQ = 0, the timer input frequency is 16 kHz (62.5 μs). If FRQ = 1, the timer input frequency is 1 kHz (1 ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

8086—8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086 = 0), A0 to A3 are used to

Table 4. MUART Registers

Read Registers								8085 Mode: AD3 AD2 AD1 AD0 8086 Mode: AD4 AD3 AD2 AD1				Write Registers							
L1	L0	S1	S0	BRKI	BITI	8086	FRQ	0	0	0	0	L1	L0	S1	S0	BRKI	BITI	8086	FRQ
Command 1												Command 1							
PEN	EP	C1	C0	B3	B2	B1	B0	0	0	0	1	PEN	EP	C1	C0	B3	B2	B1	B0
Command 2												Command 2							
0	RxE	IAE	NIE	0	SBRK	TBRK	0	0	0	1	0	SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
Command 3												Command 3							
T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0	0	0	1	1	T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
Mode												Mode							
P17	P16	P15	P14	P13	P12	P11	P10	0	1	0	0	P17	P16	P15	P14	P13	P12	P11	P10
Port 1 Control												Port 1 Control							
L7	L6	L5	L4	L3	L2	L1	L0	0	1	0	1	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Enable												Set Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	0	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Address												Reset Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Receiver Buffer												Transmitter Buffer							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Port 1												Port 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Port 2												Port 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 1												Timer 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 2												Timer 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 3												Timer 3							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 4												Timer 4							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 5												Timer 5							
INT	RBF	TBE	TRE	BD	PE	OE	FE	1	1	1	1	0	RS4	RS3	RS2	RS1	RS0	TME	DSC
Status												Modification							

address the internal registers, and an RSTn instruction is generated in response to the first $\overline{INTA}$. In 8086 mode (8086 = 1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to $\overline{INTA}$ is for 8086 interrupts where the first $\overline{INTA}$ is ignored, and an interrupt vector (40H to 47H) is placed on the bus in response to the second $\overline{INTA}$.

BITI—Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

BRKI—Break-In Detect Enable

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In. A Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1—Stop Bit Length

S1	S0	Stop Bit Length
0	0	1
0	1	1.5
1	0	2
1	1	0.75

The relationship of the number of stop bits and the function of input $\overline{CTS}$ is discussed in the Pin Description section under "$\overline{CTS}$".

L0, L1—Character Length

L1	L0	Character Length
0	0	8
0	1	7
1	0	6
1	1	5

Command Register 2

PEN	EP	C1	C0	B3	B2	B1	B0
	(1R)				(1W)		

Programming bits 0 . . . 3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0 . . . 3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0 . . . 32 Kbaud.

If bits 0 . . . 3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1.024 MHz.

B0, B1, B2, B3—Baud Rate Select

These four bits select the bit clock's source, sampling rate, and serial bit rate for the internal baud rate generator.

B3	B2	B1	B0	Baud Rate	Sampling Rate
0	0	0	0	$\overline{TxC}$, $\overline{RxC}$	1
0	0	0	1	$\overline{TxC}$/64	64
0	0	1	0	$\overline{TxC}$/32	32
0	0	1	1	19200	32
0	1	0	0	9600	64
0	1	0	1	4800	64
0	1	1	0	2400	64
0	1	1	1	1200	64
1	0	0	0	600	64
1	0	0	1	300	64
1	0	1	0	200	64
1	0	1	1	150	64
1	1	0	0	110	64
1	1	0	1	100	64
1	1	1	0	75	64
1	1	1	1	50	64

The following table gives an overview of the function of pins TxC and RxC:

Bits 3 to 0 (Hex.)	TxC	RxC
0	Input: 1 × baud rate clock for the transmitter	Input: 1 × baud rate clock for the receiver
1, 2	Input 32 × or 64 × baud rate for transmitter and receiver	Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level
3 to F	Output: baud rate clock of the transmitter	Output: as above

As an output, RxC outputs a low-to-high transition at sampling time of every data bit of a character. Thus, data can be loaded, e.g., into a shift register externally. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1—System Clock Prescaler (Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP—Even Parity (Bit 6)

EP = 0: Odd parity

EP = 1: Even parity

PEN—Parity Enable (Bit 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit

PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
	(2R)				(2W)		

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0–6 is low, no change occurs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST—Reset

If RST is set, the following events occur:

1) All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.
2) The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be canceled. Interrupt signal INT will go low.
3) The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.
4) If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does *not* alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK—Transmit Break

The transmission data output TxD will be set low as soon as the transmission of the previous character has

been finished. It stays low until TBRK is cleared. The state of $\overline{CTS}$ is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

SBRK—Single Character Break

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle (marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END—End of Interrupt

If fully nested interrupt mode is selected, this bit resets the currently served interrupt level in the Interrupt Service Register. *This command must occur at the end of each interrupt service routine during fully nested interrupt mode.* END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE—Nested Interrupt Enable

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE—Interrupt Acknowledge Enable

This bit enables an automatic response to $\overline{INTA}$. The particular response is determined by the 8086 bit in Command Register 1.

RxE—Receive Enable

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET—Bit Set/Reset

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
	(3R)				(3W)		

P2C2, P2C1, P2C0—Port 2 Control

P2C2	P2C1	P2C0	Mode	Direction Upper	Direction Lower
0	0	0	nibble	input	input
0	0	1	nibble	input	output
0	1	0	nibble	output	input
0	1	1	nibble	output	output
1	0	0	byte handshake	input	
1	0	1	byte handshake	output	
1	1	0	*DO NOT USE*		
1	1	1	test		

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3–B0 with a value ≥ 3H.

NOTE:
If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3—Counter/Timer Mode

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C—Timer 5 Control

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

T35, T24—Cascade Timers

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse. When a high-to-low transition occurs, Timer 5 is preset to its saved value, but Timer 3 is always preset to all ones. If either CT2 and CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 5.

NOTE:
Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

Table 5. Event Counters/Timers Mode of Operation

Event Counter/ Timer	Function	Programming (Mode Word)	Clock Source
1	8-Bit Timer	—	Internal Clock
2	8-Bit Timer	T24 = 0, CT2 = 0	Internal Clock
	8-Bit Event Counter	T24 = 0, CT2 = 1	P12 Pin 37
3	8-Bit Timer	T35 = 0, CT3 = 0	Internal Clock
	8-Bit Event Counter	T35 = 0, CT3 = 1	P13 Pin 36
4	8-Bit Timer	T24 = 0	Internal Clock
5	8-Bit Timer, Normal Mode	T35 = 0, T5C = 0	Internal Clock
	8-Bit Timer, Retriggerable Mode	T35 = 0, T5C = 1	Internal Clock
2 and 4 Cascaded	16-Bit Timer	T24 = 1, CT2 = 0	Internal Clock
	16-Bit Event Counter	T24 = 1, CT2 = 1	P12 Pin 37
3 and 5 Cascaded	16-Bit Timer, Normal Mode	T35 = 1, T5C = 0, CT3 = 1	Internal Clock
	16-Bit Event Counter, Normal Mode	T35 = 1, T5C = 0, CT3 = 1	P13 Pin 36
	16-Bit Timer, Retriggerable Mode	T35 = 1, T5C = 1 CT3 = 0	Internal Clock
	16-Bit Event Counter, Retriggerable Mode	T35 = 1, T5C = 1, CT3 = 1	P13 Pin 36

Port 1 Control Register

P17	P16	P15	P14	P13	P12	P11	P10
	(4R)				(4W)		

Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it is low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

Port 10, 11—Handshake Control

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as $\overline{STB}$/$\overline{ACK}$ handshake control input, and Port 11 is programmed as $\overline{IBF}$/$\overline{OBF}$ handshake control output.

If byte handshake mode is enabled for output on Port 2, $\overline{OBF}$ indicates that a character has been loaded into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving $\overline{ACK}$ low. $\overline{OBF}$ is set low by writing to Port 2 and is reset high by $\overline{ACK}$.

If byte handshake mode is enabled for input on Port 2, $\overline{STB}$ is an input. $\overline{IBF}$ is driven low after $\overline{STB}$ goes low. On the rising edge of $\overline{STB}$ the data from Port 2 is latched.

$\overline{IBF}$ is reset high when Port 2 is read.

Port 12, 13—Counter 2, 3 Input

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

Port 14—Baud Rate Generator Output Clock

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2 Kbps when it is 32 x the bit rate.

Port 15—Timer 5 Trigger

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

Port 16—Break-In Detect

If Break-In Detect is enabled by BRK1 in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

Port 17—Port Interrupt Source

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

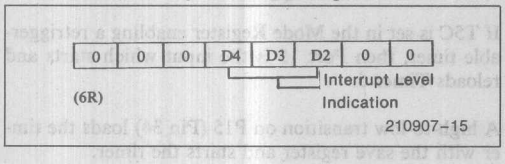
Interrupt Enable Register

L7	L6	L5	L4	L3	L2	L1	L0
	(5R)				(5W = enable, 6W = disable)		

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt Enable Register (5R).

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

Interrupt Address Register



Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge $\overline{INTA}$; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer

D7	D6	D5	D4	D3	D2	D1	D0
	(7R)				(7W)		

Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set. Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming $\overline{CTS}$ is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

D7	D6	D5	D4	D3	D2	D1	D0
	(8R)				(8W)		

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

D7	D6	D5	D4	D3	D2	D1	D0
	(9R)				(9W)		

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1–5

D7	D6	D5	D4	D3	D2	D1	D0
	($0A_{16}$–$0E_{16}$R)				($0A_{16}$–$0E_{16}$W)		

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while $\overline{RD}$ is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will unlatch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded, writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X 256+255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

INT	RBF	TBE	TRE	BD	PE	OE	FE
			($0F_{16}$R)				

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE—Framing Error, Transmission Mode

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is diabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the *first* stop bit. The error is reset by reading the Status Register or by a chip reset. A fram-

ing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is required to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

OE—Overrun Error

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

PE—Parity Error

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD—Break/Break-In

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status will be reset upon reading the status register or on a hardware of software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section under "Receive Break Detect and "Break-In Detect".

TRE—Transmit Register Empty

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If $\overline{CTS}$ is low, the Transmitter Register will be loaded during the transmission of the start bit. If $\overline{CTS}$ is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until $\overline{CTS}$ goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE—Transmitter Buffer Empty

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF—Receiver Buffer Full

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT—Interrupt Pending

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by $\overline{INTA}$ or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

0	RS4	RS3	RS2	RS1	RS0	TME	DSC

($0F_{16}W$)

DSC—Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME—Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4—Receiver Sample Time

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is *not* clocked by RxC.

NOTE:

The modification register cannot be read. Reading from address 0FH, 8086: 1EH gates the contents of the status register onto the data bus.

— A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:

- The start bit check is enabled.
- Status Register Bit 0 (FE) indicates framing error.
- The sampling time of the serial receiver is the bit center.

A software reset (Command Word 3, RST) does not affect the modification register.

RS4	RS3	RS2	RS1	RS0	Point of Time between Start of Bit and End of Bit Measured in Steps of 1/32 Bit Length
0	1	1	1	1	1 (Start of Bit)
0	1	1	1	0	2
0	1	1	0	1	3
0	1	1	0	0	4
0	1	0	1	1	5
0	1	0	1	0	6
0	1	0	0	1	7
0	1	0	0	0	8
0	0	1	1	1	9
0	0	1	1	0	10
0	0	1	0	1	11
0	0	1	0	0	12
0	0	0	1	1	13
0	0	0	1	0	14
0	0	0	0	1	15
0	0	0	0	0	16 (Bit Center)
1	1	1	1	1	17
1	1	1	1	0	18
1	1	1	0	1	19
1	1	1	0	0	20
1	1	0	1	1	21
1	1	0	1	0	22
1	1	0	0	1	23
1	1	0	0	0	24
1	0	1	1	1	25
1	0	1	1	0	26
1	0	1	0	1	27
1	0	1	0	0	28
1	0	0	1	1	29
1	0	0	1	0	30
1	0	0	0	1	31
1	0	0	0	0	32 (End of Bit)

Hardware Reset

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

1) Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.

2) Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.
3) The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT is inactive (LOW).
4) The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Big 6.
5) The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.
6) Status Register Bit 0 implies framing error.
7) The receiver samples input RxD at bit center.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 8085, 8086, 8088, and 80186 microprocessors. Figures 14 through 19 display the block diagrams for these interfaces. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0–AD3 are the address lines. For 16-bit microprocessors, AD1–AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines.

In Figure 15, the 8088 min mode, the 8205 chip select decoder is connected to the 8088's address bus lines A8–A15. These address lines are stable throughout the entire instruction cycle. However, the MUART's chip select signal could have been derived from A16/S3–A19/S6.

Figure 16 shows the 8256 interfaced with an 8086 in the min mode. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte the internal registers will be 512 address locations apart and the chip would occupy an 8K word address space. Figure 16A shows a table and a diagram of how the 8256 may be selected in an 8086 system where the MUART is I/O mapped and used on the lower byte of the address/data bus.

PROGRAMMING

Initialization

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Word 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

Command byte 1
Command byte 2
Command byte 3
Mode byte
Port 1 control
Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. It is a good idea to reset the part before initialization. (Either a hardware or a software reset will do.)

Operating the Serial Interface

The microprocessor transfers data to the serial interface by writing bytes to the Transmit Buffer Register. Receive characters are transferred by reading the Receiver Buffer Register. The Status Register provides all of the necessary information to operate the serial I/O, including when to write to the Transmit Buffer, and when to read the Receive Buffer and error information.

Transmitting

The transmitter and the receiver may be operated by using either polling or interrupts. If polling is used then the software may poll the Status Register and write a byte to the Transmit Buffer whenever TBE = 1. Writing a byte to the Transmit Buffer clears the TBE status bit. If the $\overline{CTS}$ pin is low, then the Transmit Buffer will transfer the data to the Transmit Register when it becomes empty. When this transfer takes place the TRE bit is reset, and the TBE bit is set indicating the next

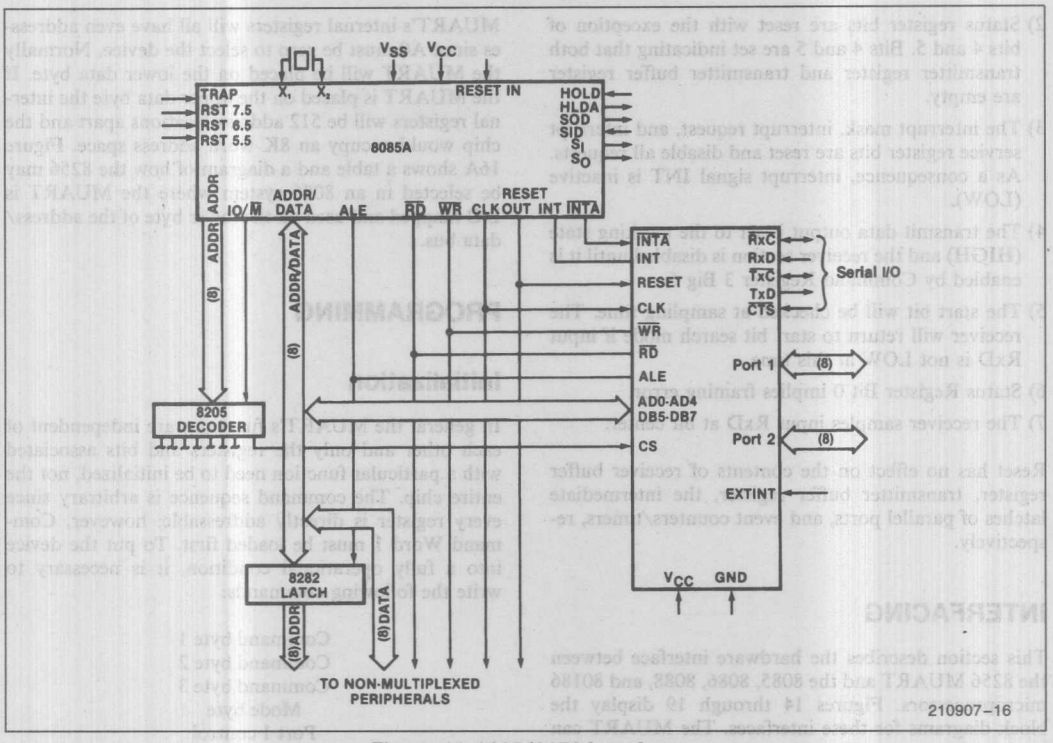


Figure 14. 8085/8256 Interface

byte may be written to the Transmit Buffer. If $\overline{CTS}$ is high, disabling the transmitter, the data byte will remain in the Transmit Buffer and TBE will remain low until $\overline{CTS}$ goes low. The transmitter can only buffer one byte if it is disabled.

There is no way of knowing that the transmitter is disabled unless the $\overline{CTS}$ signal is fed into one of the I/O ports. Using the transmitter interrupt will free up the CPU to perform other functions while the transmitter is disabled or while the Transmit Buffer is full.

To enable the transmit interrupt feature Bit L5 in the Set Interrupt Register must be set. An interrupt request will not occur immediately after this bit has been set. Before any transmit interrupt request will occur a byte must be written to the Transmit Buffer. After the first byte has been written to the Transmit Buffer, a transmit interrupt request will occur, providing the transmitter is enabled.

There are three sources of transmitter interrupt requests: TBE = 1, TRE = 1, and Break-In Detect. Assuming the Break-In Detect feature is disabled, after the transmit interrupt is enabled and the first byte is written, a transmit interrupt request will be generated by TBE going active. The microprocessor can immediately write a byte to the Transmit Buffer without reading any status. However if Break-In Detect is enabled, the Status Register must be read to determine whether the transmit interrupt request was generated by Break-In Detect or TBE.

The TRE interrupt request can be used to indicate when the transmitter has completely sent all of the data. For example, using half-duplex communications, all of the data written to the MUART must be transmitted before the line can be turned around. After the last byte is written, an interrupt request will be generated by TBE. If this interrupt is acknowledged without writing another byte, then the next transmitter interrupt request, TRE = 1, will indicate that the transmitter is empty and the line may be turned around.

RECEIVING

Valid data may be read from the Receive Buffer whenever the RBF bit in the Status Register is set. Reading

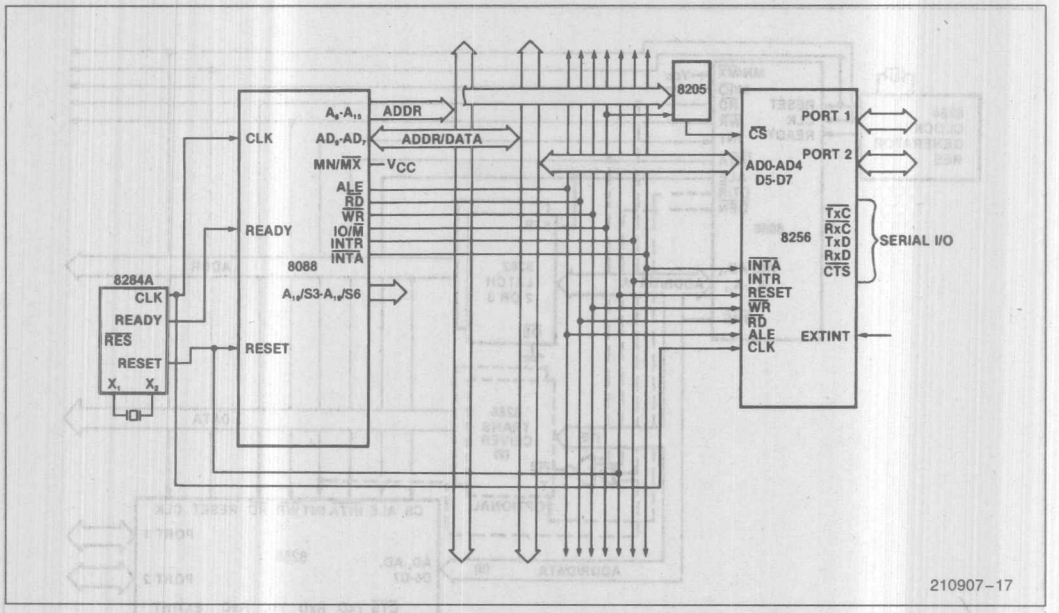


Figure 15. 8088 Min Mode/8256 Interface Multiplexed Bus

the Receive Buffer resets the RBF status bit. The RBF bit in the Status Register can be used for polling. When the RBF bit is set, the three receive status bits, PE, OE, and FE are updated. These three status bits are reset when they are read. Therefore when the status register is read with RBF set, the three error status bit should be tested too.

If interrupts are used for serial receive data, the receiver must be enabled by setting the RxE bit in Command Register 3, and Bit L4 must be set in the Set Interrupt Register. When the receive interrupt request occurs the Receive Buffer may be read, but the status register should also be read since the receive interrupt could have been generated by the Break Detect. Also, reading the status register will indicate whether there were any errors in the received character.

Operating the Parallel Interface

Data can be transferred to or read from Port 1 and Port 2 by using the appropriate write and read operations.

LOADING PORT 1 AND PORT 2

Writing to the ports transfers the data present on the data bus into the output latches. This operation is independent of the programmed I/O characteristics of the individual port pins. Writing to control or input ports has no effect on the state of the pins. Pins defined as outputs immediately assume the state which is associated with the transferred data. If inputs or control pins are reprogrammed into outputs, they assume the states stored in their output latches which were transferred by the most recent port write operation.

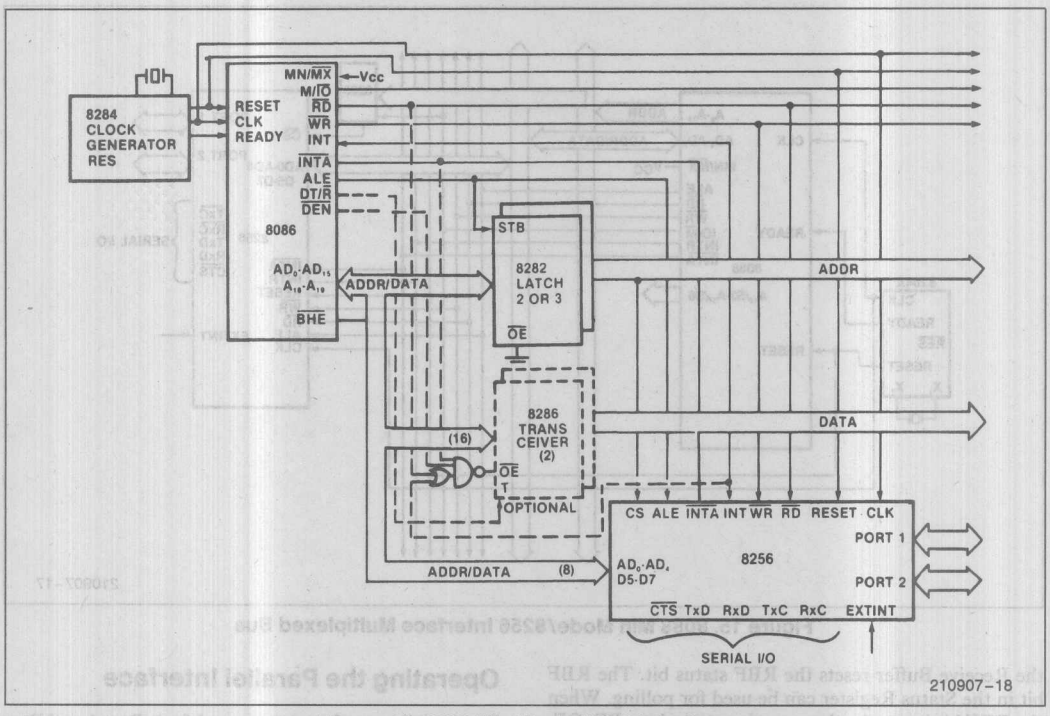
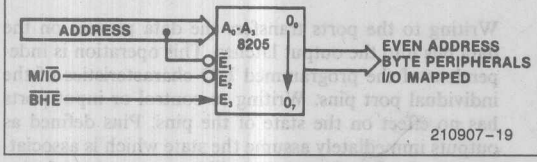


Figure 16. 8086 Min Mode/8256 Interface



MUART →

$\overline{BHE}$	A_0	Characteristics
0	0	Whole Word
0	1	Upper Byte from/to Odd Address
1	0	Lower Byte from/to Even Address
1	1	None

Figure 16a. Technique for Generating the MUART's Chip Select

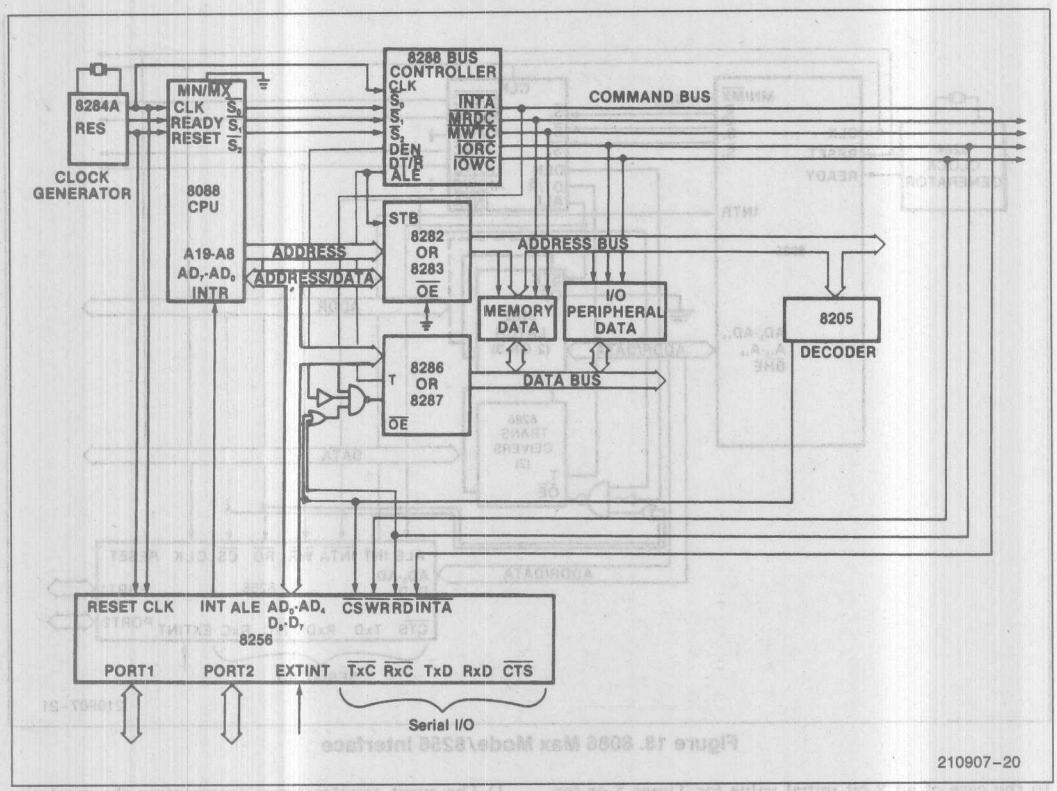


Figure 17. 8088 Max Mode/8256 Interface

READING PORT 1 AND PORT 2

Reading the ports gates the state at the pins onto the data bus if they are defined as I/O pins. A read operation transfers the contents of the associated output latches of pins P12, P13, P15, and P16, which are defined as control function pins. Reading control pins P10, P11, and P17 delivers the state of these pins.

Operating the Event Counters/Timers

The event counters/timers can be loaded with an intial value at any time. Reading event counters/timers is possible without interfering with the counting process.

LOADING EVENT COUNTERS/TIMERS

Loading event counters/timers 1–5 under their respective addresses transfers the data present on the data bus as an initial value into the addressed event counter/timer. The event counter/timer counts from the new initial value immediately following the data transfer (exception: retriggerable mode of Timer 5, or 3 and 5).

Cascaded counters/timers can be loaded with an initial value using one of two procedures:

1) Only the event counter/timer representing the most significant byte will be loaded. The event counter/timer representing the least significant byte is set to 0FFH automatically. Counting is started immediately after the data transfer.
2) The event counter/timer representing the most significant byte will be loaded, causing the least significant byte to be set to 0FFH automatically. Counting is started immediately following the data transfer. Next, the counter representing the least significant byte will be loaded and counting is started again, but this time with a complete 16-bit initial value. The least significant byte of the initial value must be transferred before the counter representing the least significant byte exhibits its zero transition to prevent the most significant byte of the initial value from being decremented improperly.

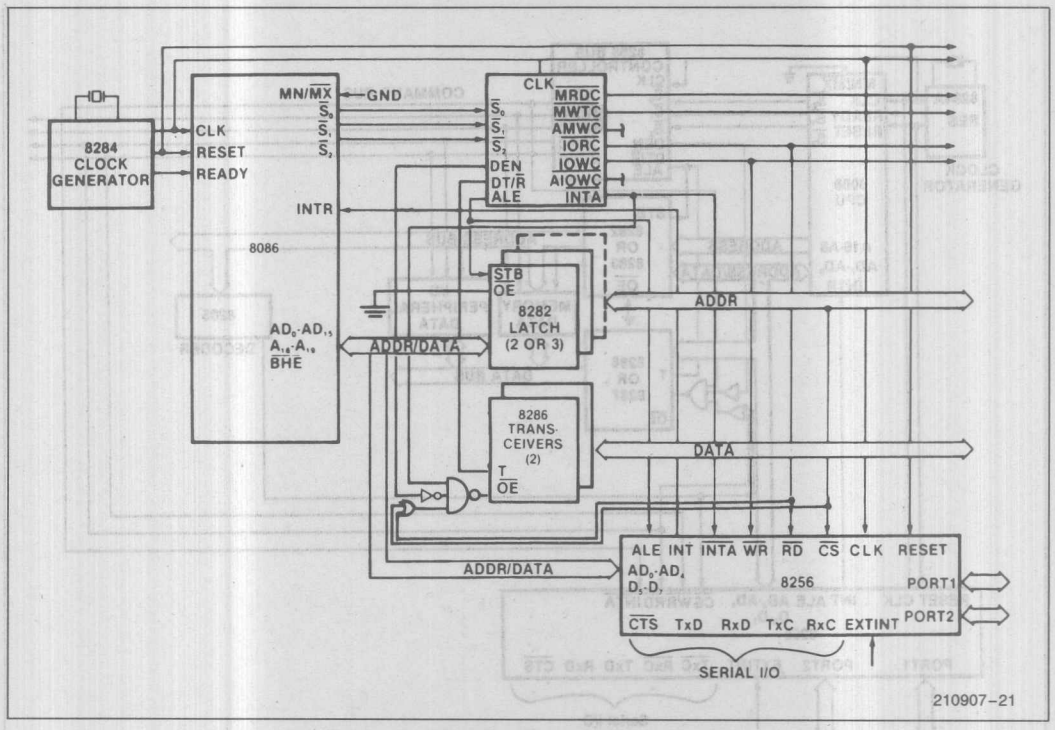


Figure 18. 8086 Max Mode/8256 Interface

In the case of an 8-bit initial value for Timer 5 or for cascaded Event Counter/Timer 3 and 5, the initial value for Timer 5 is loaded from a save register, if it is operated in retriggerable counting mode. Counting is started after an initial value has been transferred whenever a high-to-low transition occurs on Port P15.

Cascaded Event Counter/Timer 3 and 5 operating in retriggerable counting mode can be loaded directly with an initial value for Timer 5 representing the most significant byte; Event Counter/Timer 3 will be set to 0FFH automatically.

READING EVENT COUNTERS/TIMERS

Reading event counters/timers 1–5 from their respective addresses gates the counter contents onto the data bus. The counter contents gated onto the data bus remain stable during the read operation while the counter just being read continues to count. The minimum time between the two read operations from the same counter is 1 μs.

The procedure to be followed when reading cascaded event counters/timers is:

1) The event counter/timer representing the most significant byte will be read first. At this time, the least significant byte is latched onto read latches.
2) When the event counter/timer representing the least significant byte is addressed, the byte stored in the read latches will be gated onto the data bus. The value stored in the read latches remains valid until it is read, the cascading condition is removed, or a write operation affecting one of the two event counters/timers is executed.

The time between reading the most significant byte and the least significant byte must be a least 1 μs.

NOTE:

For cascaded event counters/timers the least significant counter/timer is latched after reading the most significant counter/timer. If the lower byte changes from 00H to 0FFH between the reading of the MSB and the latching of the LSB, the carry from the most significant event counter/timer to the least significant event counter/timer is lost. Therefore, it is necessary to repeat the whole reading once if the value of the least significant event counter/timer is 0FFH. Doing this will avoid working with a wrong value (correct value + 255).

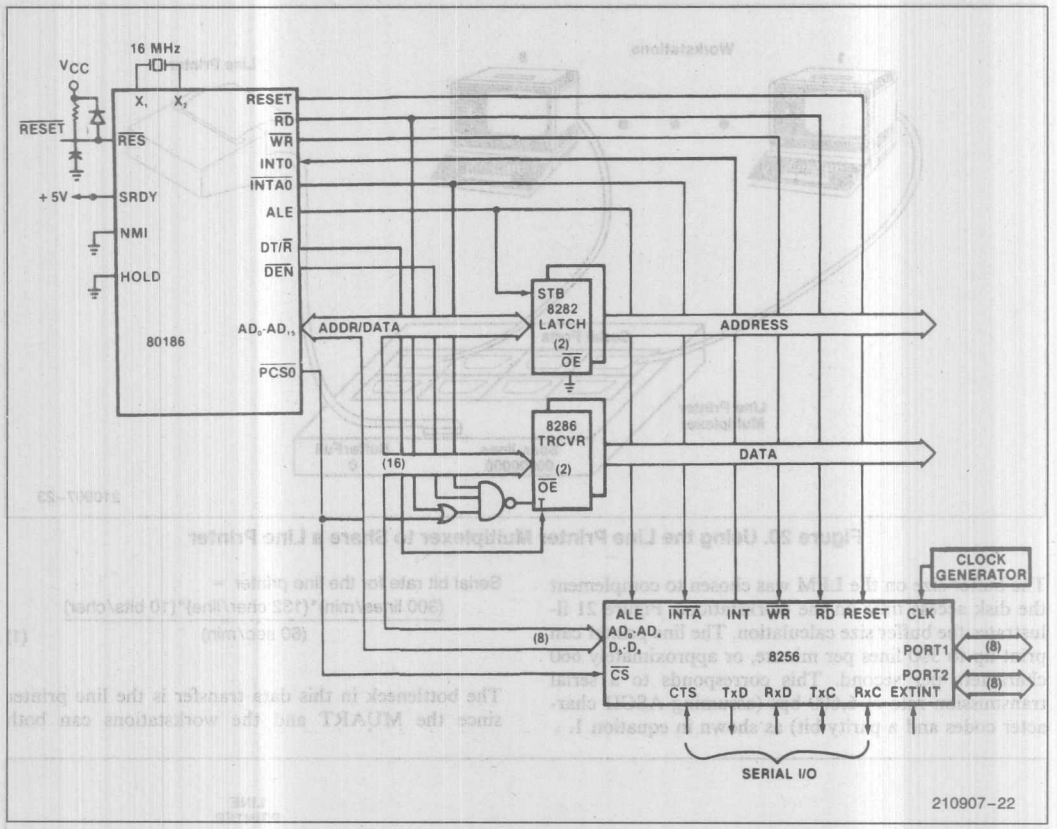


Figure 19. 80186/8256 Interface

APPLICATION EXAMPLE

This section describes how the 8256 was designed into a Line Printer Multiplexer (LPM). This application example was chosen because it employs a majority of the MUART's features. The information in this section will be applicable to many other designs since it describes some common software and hardware aspects of using the MUART.

Description of the Line Printer Multiplexer (LPM)

The Line Printer Multiplexer allows up to eight workstations to share one printer. The workstations transmit serial asynchronous data to the LPM. The LPM receives the serial data, buffers it, then transmits it to the line printer using a two-wire byte handshake Dataproducts interface. A conceptual diagram of this system is shown in Figure 20. Note that only one workstation can transmit at a time. This workstation will transmit its entire file before another workstation will be allowed to transmit.

The LPM sequentially polls each of the eight RS-232 ports for a Request To Send ($\overline{RTS}$). When it finds a serial port which has asserted $\overline{RTS}$, it configures itself for the appropriate data format and bit rate, establishes the connection and sends back to the serial port a Clear To Send ($\overline{CTS}$) which enables transmission. The LPM receives the serial asynchronous data, buffers it in a software FIFO, and transmits the data to the line printer. If the LPM detects an error in any of the serial characters it receives, it transmits an error message to the serial port and ignores the bad character. If the LPM does not receive a serial character after 18 seconds, it assumes that the transmission is complete. It transmits the final status to the serial port, and returns to scanning.

This LPM was designed to be used with single-user workstations and a 300 lines per minute line printer. These workstations are not multitasking; therefore in the middle of a file transfer when the CPU needs to reload its buffer from the disk, no serial data is transmitted. During this time the LPM is emptying its FIFO; thus, the line printer never stops printing.

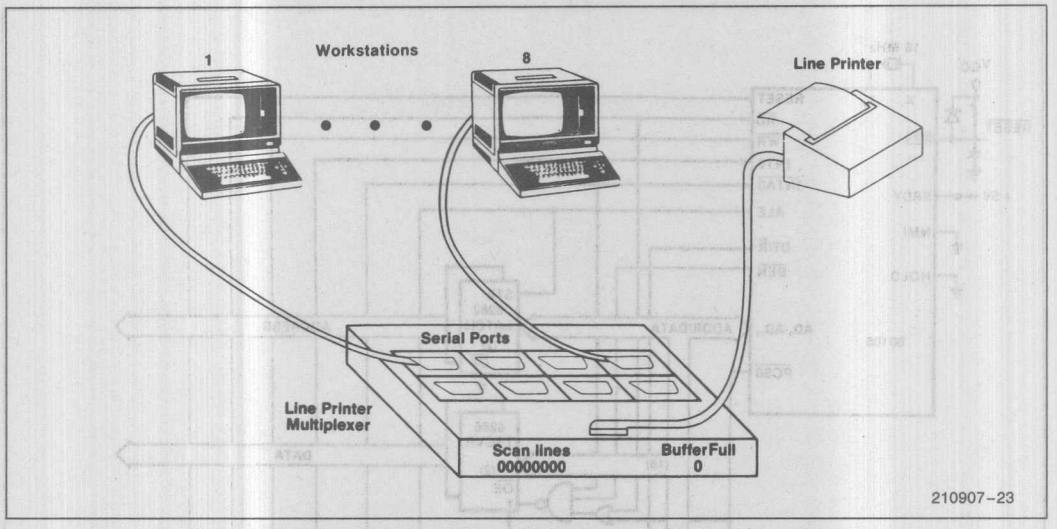


Figure 20. Using the Line Printer Multiplexer to Share a Line Printer

The buffer size on the LPM was chosen to complement the disk access time on the workstations. Figure 21 illustrates the buffer size calculation. The line printer can print up to 300 lines per minute, or approximately 660 characters per second. This corresponds to a serial transmission rate of 6,600 bps (assuming ASCII character codes and a parity bit) as shown in equation 1.

$$\text{Serial bit rate for the line printer} = \frac{(300\ \text{lines/min})*(132\ \text{char/line})*(10\ \text{bits/char})}{(60\ \text{sec/min})} \quad (1)$$

The bottleneck in this data transfer is the line printer since the MUART and the workstations can both

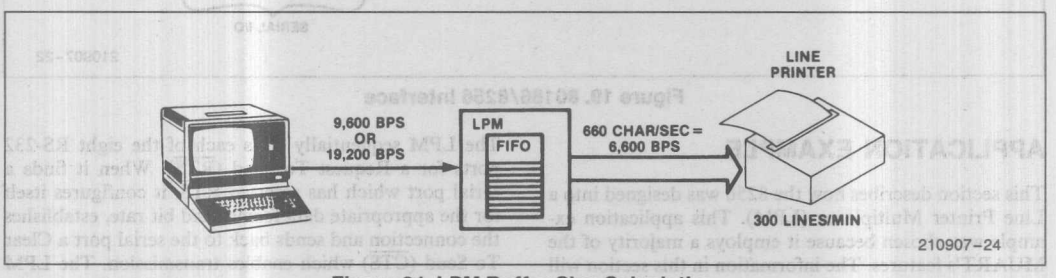


Figure 21. LPM Buffer Size Calculation

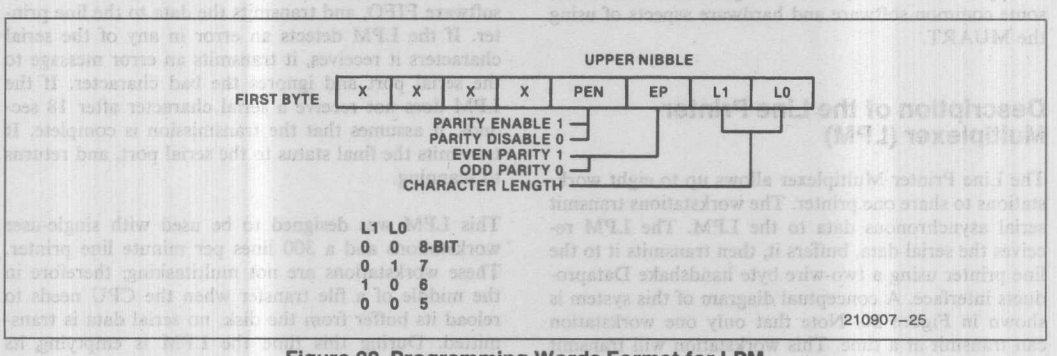


Figure 22. Programming Words Format for LPM

transmit and receive at 19.2 Kbps. To realize the maximum data transfer rate of this system the LPM must guarantee that the average transfer rate to the line printer is 660 characters per second. The maximum amount of dead time that the serial port on the workstation is not transmitting, multiplied by 660 is the number of bytes which the LPM should buffer. It was determined through experimentation that it takes about 3 seconds to load 40K bytes of data from the disk into the workstation's RAM. During these 3 seconds no serial data is being sent; therfore the buffer size on the LPM should be 2K bytes. (Note: even though only a 2K byte FIFO is required, this design used an 8K byte FIFO.)

To keep the LPM's buffer full the serial data rate must be greater than 6.6 Kbps. The two bit rates which the workstations use are 9.6 Kbps and 19.2 Kbps. The $\overline{CTS}$ signal is used to control the flow of the serial data so that the LPM buffer will not overflow.

Each serial port on the LPM can have a different bit rate, character length, and parity format. These parameters are programmable through the serial port. When the LPM powers up, or is reset, it expects a bit rate of 9600 bps, 7 bit characters, and odd parity. When a serial port receives an ASCII ESC character (1BH), it puts that port in the program mode. The next two bytes will program these three parameters. Only the lower nibbles of these two bytes are used, and the upper nibbles are discarded. The format of these programming words is given in Figure 22. If the word following the ESC is an ASCII NUL (0), the LPM will exit from the programming mode and not change any of its parameters.

Description of the Hardware

Figure 23 shows a block diagram of the LPM. In addition to the standard components of most microprocessor systems such as CPU, RAM, and ROM this particular design requires a UART, timers, parallel I/O and an interrupt controller. The MUART is the ideal choice for this design since it integrates these four functions onto one device.

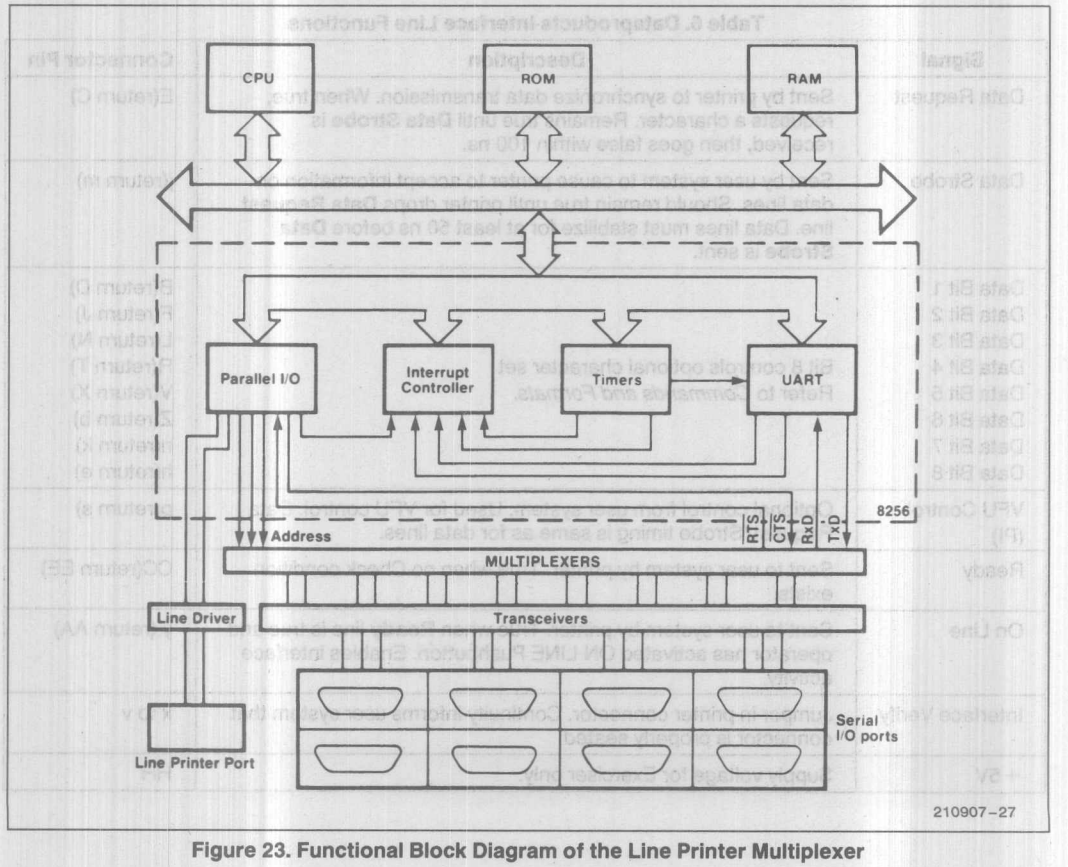


Figure 23. Functional Block Diagram of the Line Printer Multiplexer

The eight serial I/O ports use four signals: Transmit Data (TxD), Receive Data (RxD), Request To Send ($\overline{RTS}$), and Clear To Send ($\overline{CTS}$). These four signals, controlled by the MUART, are connected to one port at a time using TTL multiplexers. The TTL multiplexers are interfaced to RS-232 transceivers to be electrically compatible with the RS-232 spec. The serial port select address is derived from three bits of the MUART's parallel I/O port (Port 1). Two more bits from Port 1 control $\overline{CTS}$ and $\overline{RTS}$, and another bit lights up an LED to indicate when the LPM's buffer is full. Parallel Port 2 and two bits from Port 1 are connected to the line printer implementing a two-wire byte handshake transfer. These signals are passed through a line driver so that they can reliably drive a long cable.

There are three timing functions needed for the LPM: a scan timer, a debounce timer, and a receive timeout. The Scan timer determines the amount of time spent sampling $\overline{RTS}$ on each port before the next port is addressed. By using one of the MUART's timers to do this function, the CPU is free to perform other functions instead of implementing the timer in software. If $\overline{RTS}$ is recognized as true, the CPU branches into a debounce procedure. This procedure uses another one of the MUART's timers to wait 10 ms then sample $\overline{RTS}$ again, thus preventing any glitches from registering as a false $\overline{RTS}$. The receive timeout timer uses two 8-bit timers in the cascaded mode to measure an 18-second interval. After a valid $\overline{RTS}$ is recognized, the LPM sends back a $\overline{CTS}$ and initializes the receive timeout timer for 18 seconds. Each time a character is received by the LPM, this timer is reinitialized. If this timer times out, the LPM considers the transmission complete and returns to scanning.

The schematic diagram of the LPM is shown in Figure 24. The CPU is an 8088 used in the min mode. It is interfaced directly to the 8256. An 8282 latch is employed in the system so that nonmultiplexed bus memory can be used. A 2716 holds the entire program, and six 2016s (2K x 8 static RAMs) are used to store the buffer, temporary data, stack area, and interrupt vector table. The 2716 is located in the upper 2K of the 8088

Table 6. Dataproducts Interface Line Functions

Signal	Description	Connector Pin
Data Request	Sent by printer to synchronize data transmission. When true, requests a character. Remains true until **Data Strobe** is received, then goes false within 100 ns.	E(return C)
Data Strobe	Sent by user system to cause printer to accept information on data lines. Should remain true until printer drops **Data Request** line. Data lines must stabilize for at least 50 ns before **Data Strobe** is sent.	j(return m)
Data Bit 1 Data Bit 2 Data Bit 3 Data Bit 4 Data Bit 5 Data Bit 6 Data Bit 7 Data Bit 8	Bit 8 controls optional character set Refer to *Commands and Formats.*	B(return D) F(return J) L(return N) R(return T) V(return X) Z(return b) n(return k) h(return e)
VFU Control (PI)	Optional control from user system. Used for VFU control. Data Request/Strobe timing is same as for data lines.	p(return s)
Ready	Sent to user system by printer. True when no Check condition exists.	CC(return EE)
On Line	Sent to user system by printer. True when **Ready** line is true and operator has activated ON LINE Pushbutton. Enables interface activity.	y (return AA)
Interface Verify	Jumper in printer connector. Continuity informs user system that connector is properly seated.	x to v
+5V	Supply voltage for Exerciser only.	HH

address space (FF800-FFFFFH) so that the reset vectors can be stored starting at location FFFF0H. The RAM address space spans 0-2FFFH so that the interrupt vector table can be stored starting at location 0. The MUART is I/O mapped and its registers occupy even addresses from 0 to 1EH. Using an 8088 CPU the MUART must be placed in the 8086 mode since the $\overline{\text{INTA}}$ signal is used; hence the register addresses are all even numbers.

The line printer used provides a choice of two standard parallel interfaces: Centronics or Dataproducts. The Centronics interface uses a two-wire handshake pulsed strobe where the transmitter asserts a complete strobe pulse before an acknowledge is received. The Dataproducts interface is an interlocking two-wire handshake. The Dataproducts interface was chosen since it is directly compatible with the MUART's two-wire byte handshake. The MUART could also be connected to the Centronics interface; however, additional hardware would be necessary to generate the pulsed strobe for correct interrupt operation. Figure 25 shows the timing of the Dataproducts interface and Table 6 lists the connector pin configuration.

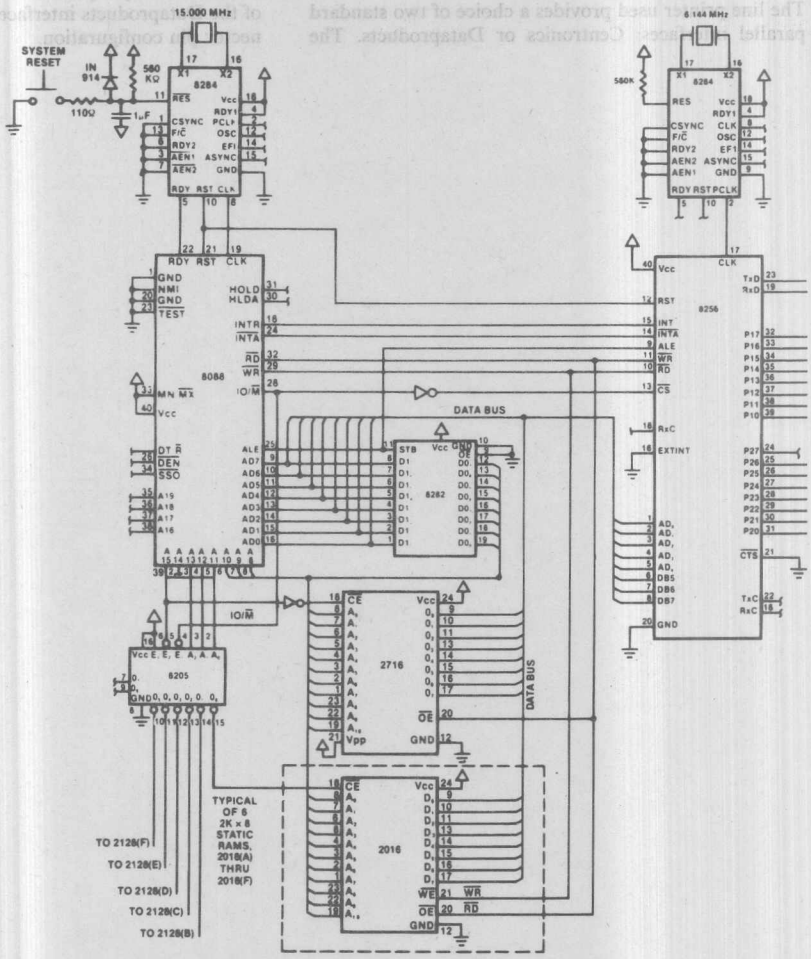


Figure 24. Schematic of LPM

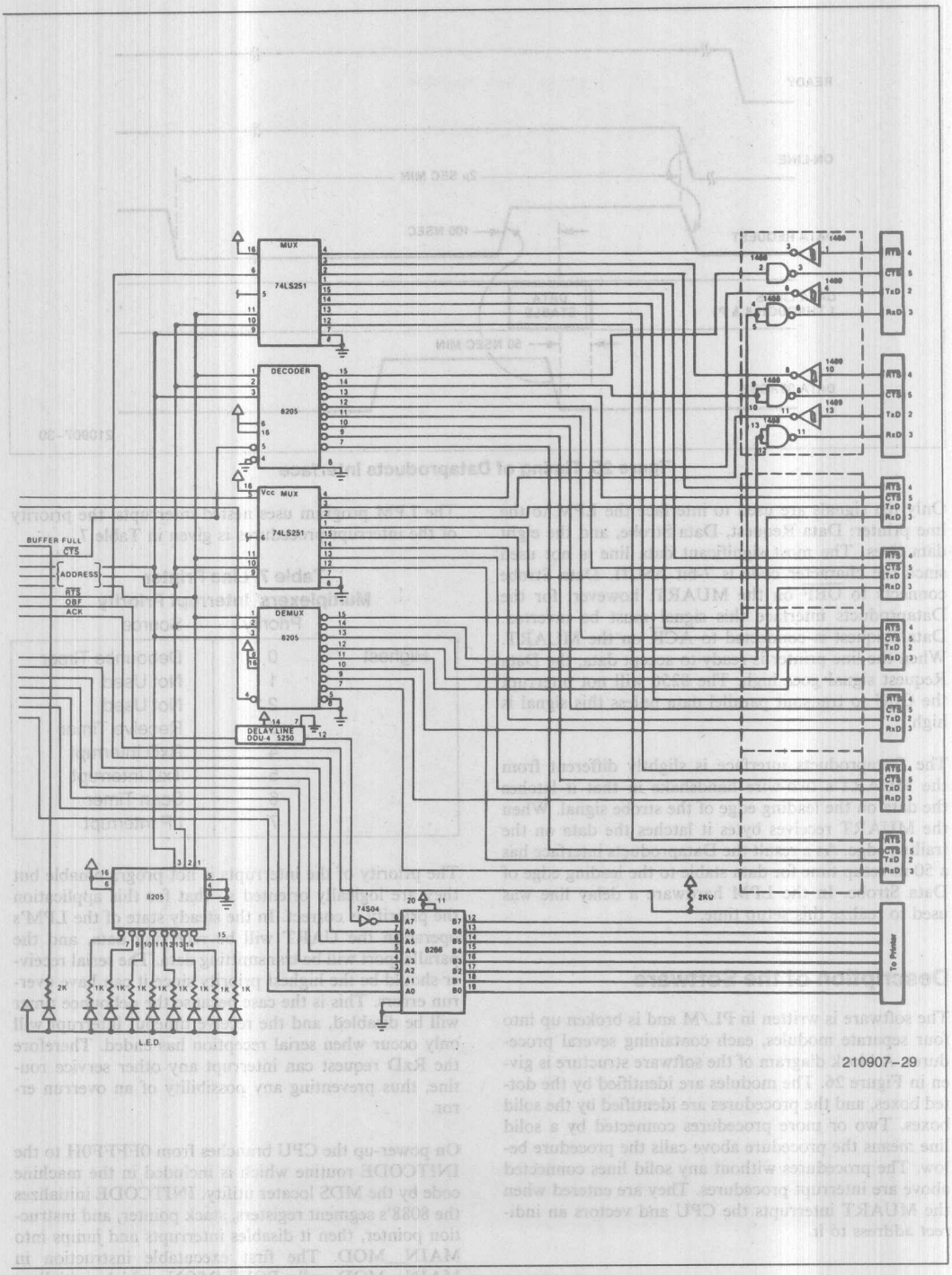


Figure 24. Schematic of LPM (Continued)

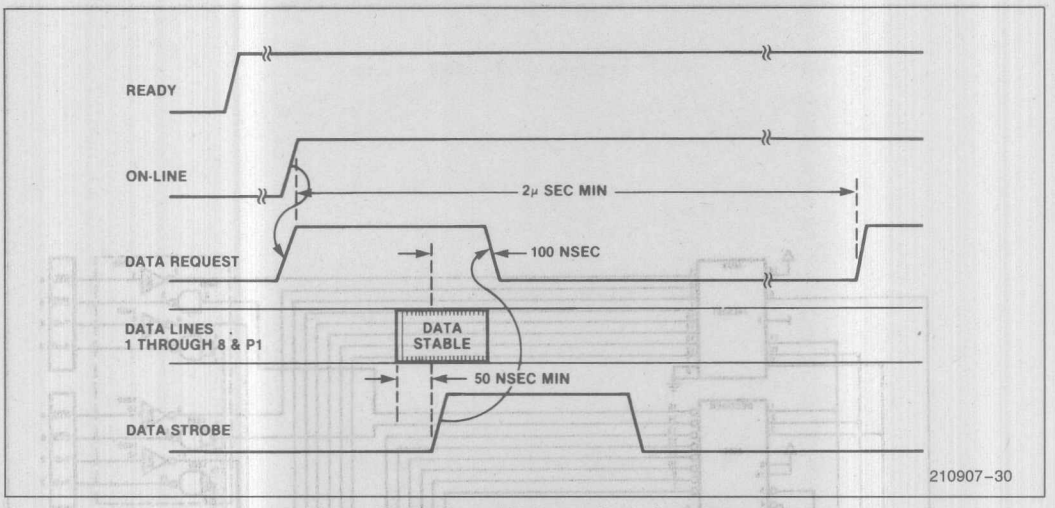


Figure 25. Timing of Dataproducts Interface

Only ten signals are used to interface the LPM to the line printer: Data Request, Data Strobe, and the eight data lines. The most significant data line is not used since the character code is 7-bit ASCII. Data Strobe connects to $\overline{OBF}$ on the MUART; however, for the Dataproducts interface this signal must be inverted. Data Request is connected to $\overline{ACK}$ on the MUART. When the line printer is ready to accept data, the Data Request signal goes high. The 8256 will not interrupt the CPU to transmit parallel data unless this signal is high.

The Dataproducts interface is slightly different from the MUART's two-wire handshake in that it latches the data on the leading edge of the strobe signal. When the MUART receives bytes it latches the data on the trailing edge. As a result the Dataproducts interface has a 50 ns setup time for data stable to the leading edge of Data Strobe. In the LPM hardware a delay line was used to realize this setup time.

Description of the Software

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 26. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connected above are interrupt procedures. They are entered when the MUART interrupts the CPU and vectors an indirect address to it.

The LPM program uses nested interrupts; the priority of the interrupt procedures is given in Table 7.

Table 7. Line Printer Multiplexers' Interrupt Priority

	Priority	Source
Highest	0	Debounce Timer
	1	Not Used
	2	Not Used
	3	Receive Timer
	4	RxD Interrupt
	5	TxD Interrupt
	6	Scan Timer
	7	LP Interrupt

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. In the steady state of the LPM's operation the UART will be receiving data, and the parallel port will be transmitting data. The serial receiver should be the highest priority since it can have overrun errors. This is the case because the debounce timer will be disabled, and the receive timeout interrupt will only occur when serial reception has ended. Therefore the RxD request can interrupt any other service routine, thus preventing any possibility of an overrun error.

On power-up the CPU branches from 0FFFF0H to the INITCODE routine which is included in the machine code by the MDS locater utility. INITCODE initializes the 8088's segment registers, stack pointer, and instruction pointer, then it disables interrupts and jumps into MAIN_MOD. The first executable instruction in MAIN_MOD calls POWER$ON, which initializes

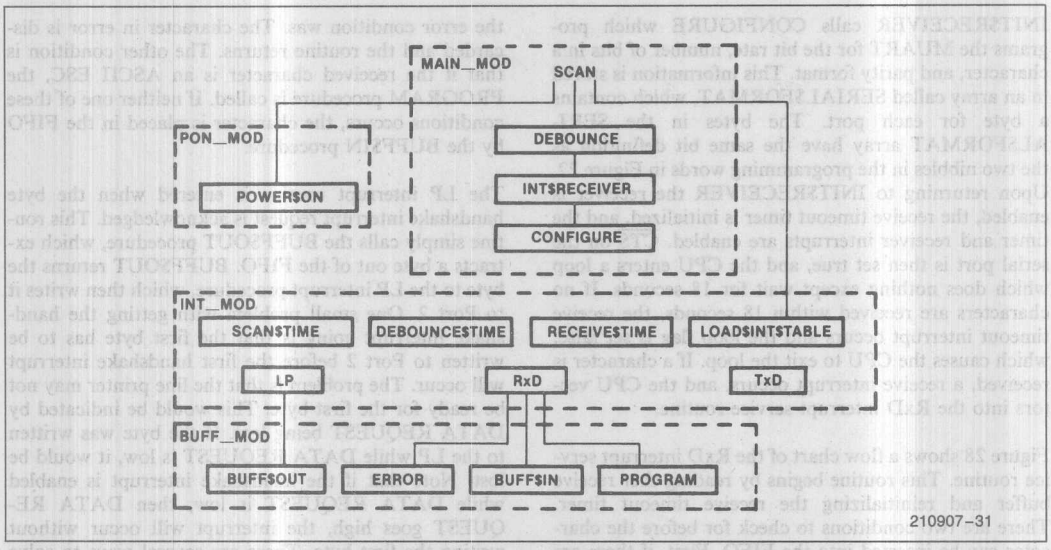


Figure 26. Block Diagram of LPM Software Structure

the MUART, flags, variables, and arrays. The MAIN_MOD calls LOADINTTABLE, which initializes the interrupt vector table. The CPU's interrupt is then enabled and the program enters into a DO FOREVER loop which scans the eight serial ports for an $\overline{RTS}$.

There are three software functions which employ the MUART's timers and nterrupt controller to measure time intervals: SCAN, debounce, and INIT$RECEIVER. DEBOUNCE and INIT$RECEIVER procedures, employ the MUART's timers and interrupt controller to measure time intervals. The CPU remains in a loop for a specific amount of time before it proceeds with the next section of code. In this loop the CPU is waiting for a global status flag to change while servicing any interrupts which may occur. When the appropriate timer interrupt occurs, the interrupt service routine will set the global flag which causes the CPU to exit the loop and proceed to the next section of code. An example can be seen from the scan flow chart in Figure 27.

The first thing the program does before entering the loop is set the flag (in this case SCAN$DELAY) TRUE. The timer is intialized and the loop is entered. As long as SCAN$DELAY is TRUE the CPU will continue to sample $\overline{RTS}$. If $\overline{RTS}$ remains false for more than 100 ms, the timer interrupts the CPU and the interrupt service routine sets SCAN$DELAY FALSE This causes the CPU to exit the loop and address the next port. The process is then repeated. If $\overline{RTS}$ becomes true while it is being sampled, the DEBOUNCE procedure is called.

DEBOUNCE does nothing more than wait 10 ms and sample $\overline{RTS}$ again using the same technique discussed above. If $\overline{RTS}$ is still valid INIT$RECEIVER is called, otherwise the CPU returns to scan.

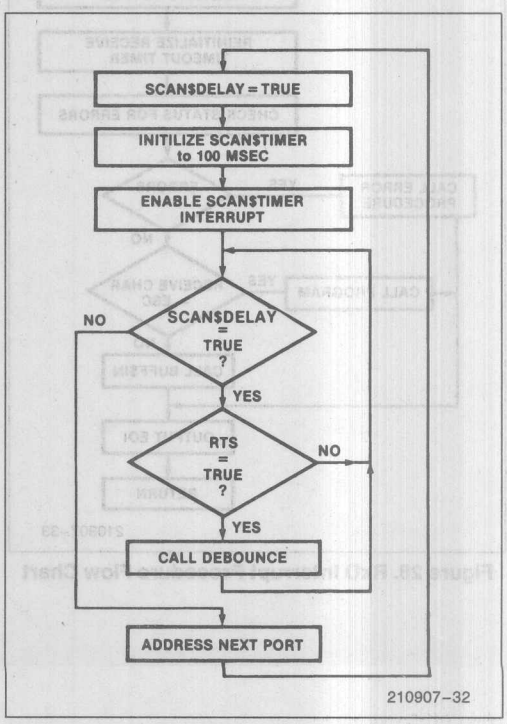


Figure 27. Scan Flow Chart

INIT$RECEIVER calls CONFIGURE which programs the MUART for the bit rate, number of bits in a character, and parity format. This information is stored in an array called SERIAL$FORMAT, which contains a byte for each port. The bytes in the SERIAL$FORMAT array have the same bit definition as the two nibbles in the programming words in Figure 22. Upon returning to INIT$RECEIVER the receiver is enabled, the receive timeout timer is initialized, and the timer and receiver interrupts are enabled. $\overline{CTS}$ on the serial port is then set true, and the CPU enters a loop which does nothing except wait for 18 seconds. If no characters are received within 18 seconds, the receive timeout interrupt occurs and the loop flag is set false, which causes the CPU to exit the loop. If a character is received, a receive interrupt occurs, and the CPU vectors into the RxD interrupt service routine.

Figure 28 shows a flow chart of the RxD interrupt service routine. This routine begins by reading ther receive buffer and reinitializing the receive timeout timer. There are two conditions to check for before the character can be inserted into the FIFO. First, if there are any errors in the received character, an ERROR procedure is called which reports back to the serial port what

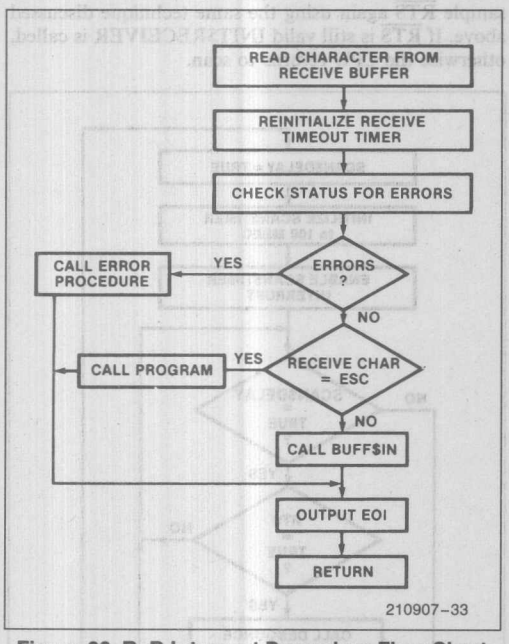


Figure 28. RxD Interrupt Procedure Flow Chart

the error condition was. The character in error is discarded and the routine returns. The other condition is that if the received character is an ASCII ESC, the PROGRAM procedure is called. If neither one of these conditions occurs, the character is placed in the FIFO by the BUFF$IN procedure.

The LP interrupt routine is entered when the byte handshake interrupt request is acknowledged. This routine simply calls the BUFF$OUT procedure, which extracts a byte out of the FIFO. BUFF$OUT returns the byte to the LP interrupt procedure, which then writes it to Port 2. One small problem with getting the handshake interrupt going is that the first byte has to be written to Port 2 before the first handshake interrupt will occur. The problem is that the line printer may not be ready for the first byte. This would be indicated by DATA REQUEST being low. If the byte was written to the LP while DATA REQUEST is low, it would be lost. Note that if the handshake interrupt is enabled while DATA REQUEST is low, then DATA REQUEST goes high, the interrupt will occur without writing the first byte. There are several ways to solve this problem. Port 1 can be read to find out what the state of the DATA REQUEST line is. If DATA REQUEST is low, the CPU can simply wait for the interrupt without writing the first byte. If DATA REQUEST is high, then the first data byte may be written. Another solution would be to write a NUL character as the first byte to Port 2. If DATA REQUEST is low, then a worthless character is lost. If DATA REQUEST is high, the NUL character would be sent to the line printer; however, it is not printed since NUL is a non-printable character. The LPM program uses the NUL character solution.

BUFFER MANAGEMENT

The FIFO implementation uses an 8K byte array to store the characters. There are two pointers used as indexes in the array to address the characters: IN$POINTER and OUT$POINTER. IN$POINTER points to the location in the array which will store the next byte of data inserted. OUT$POINTER points to the next byte of data which will be removed from the array. Both IN$POINTER and OUT$POINTER are declared as words. Figure 29 illustrates the FIFO in a block diagram.

The BUFF$IN procedure receives a byte from the RxD interrupt routine and stores it in the array location pointed to by IN$POINTER, then IN$POINTER is incremented. Similarly, when BUFF$OUT is called by the LP interrupt routine, the byte in the array pointed

to by OUT$POINTER is read. OUT$POINTER is incremented, and the byte which was read is passed back to the LP interrupt routine. Since IN$POINTER and OUT$POINTER are always incremented, they must be able to roll over when they hit the top of the 8K byte address space. This is done by clearing the upper three bits of each pointer after it is incremented.

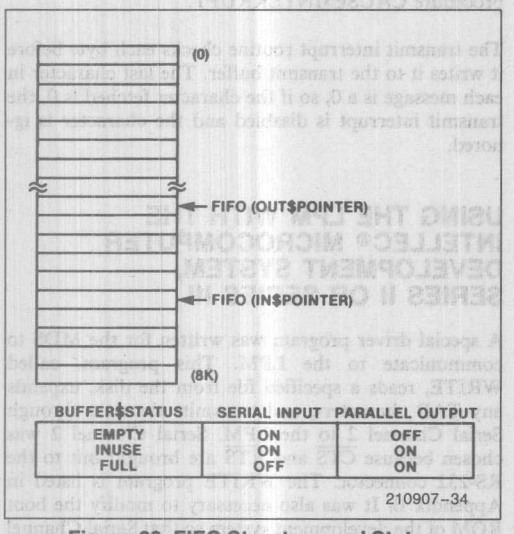


Figure 29. FIFO Structure and Status

IN$POINTER and OUT$POINTER not only point to the locations in the FIFO, they also indicate how many bytes are in the FIFO and whether the FIFO is full or empty. When a character is placed into the FIFO and IN$POINTER is incremented, the FIFO is full if IN$POINTER equals OUT$POINTER. When a character is read from the FIFO and OUT$POINTER is incremented, the FIFO is empty if OUT$POINTER equals IN$POINTER. If the buffer is neither full nor empty, then it is in use. A byte called BUFFER$STATUS is used to indicate one of these three conditions.

The software uses the buffer status information to control the flow into and out of the FIFO. When the FIFO is empty the handshake interrupt must be turned off. When the FIFO is full, $\overline{CTS}$ must be sent false so that no more data will be received. If the buffer status is in use, $\overline{CTS}$ is true and the handshake interrupt is enabled.

Figure 30 shows the flow chart of the BUFF$IN procedure. The BUFF$IN procedure begins by checking the BUFFER$STATUS. If it is empty and the character to be inserted into the FIFO is a CR or LF, the handshake interrupt is enabled, a NUL character is output, and the BUFFER$STATUS is set to INUSE. The character passed to BUFF$IN from RxD is put into the FIFO. If the FIFO is now full, the BUFFER$STATUS is set to FULL, $\overline{CTS}$ is set false, and the buffer full LED is turned on.

Figure 31 shows the flow chart of the BUFF$OUT procedure. After the character is read from the FIFO, the FIFO is tested to determine if it is empty. If it is not empty, the BUFFER$STATUS is FULL and there are 200 bytes available in the FIFO, serial data reception is reenabled, and the FIFO fills again. White data is being received from the workstation, $\overline{CTS}$ toggles high and low, filling up and emptying the last 200 bytes in the FIFO. Referring to the top of the flow chart (FIFO empty test) if it's empty, the BUFFER$STATUS is set to EMPTY, and the handshake interrupt is disabled. During this time all interrupts are disabled at the CPU. (Remember that the RxD interrupt routine can interrupt the LP and BUFF$OUT procedures since it has a higher priority, and the MUART is in the nested mode.)

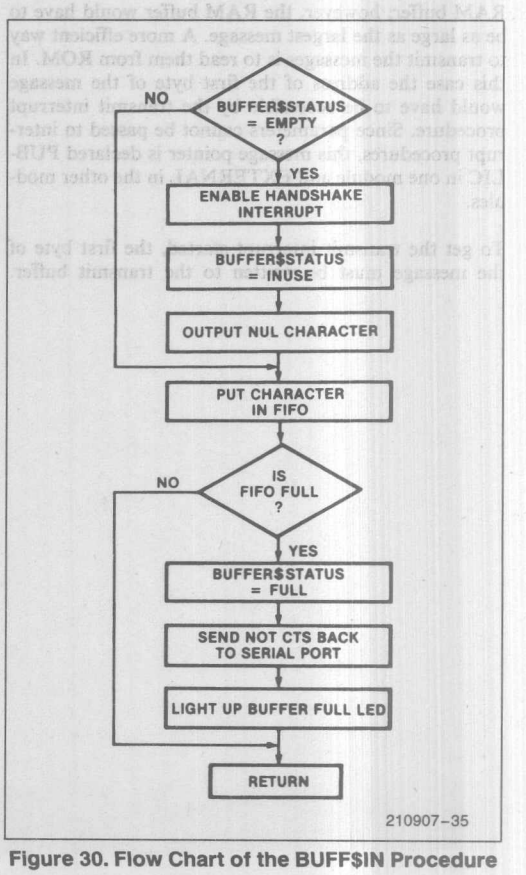


Figure 30. Flow Chart of the BUFF$IN Procedure

If the CPU interrupt was not disabled during this time, the following events could occur which would cause the LPM to crash. Assume that the RxD interrupt occurred where the asterisk is in the flow chart, after BUFFER$STATUS is set to EMPTY. The BUFF$IN procedure would set BUFFER$STATUS to INUSE and enable the handshake interrupt. When the RxD interrupt routine returned to BUFF$OUT, the handshake interrupt is disabled, but the BUFFER$STATUS is INUSE. The handshake interrupt could never be reenabled, and the FIFO would fill up. This is known as a critical section of code. Suspicion should arise for a critical section of code when two or more nested interrupt routines can affect the same status. One solution is to disable the interrupt flag at the CPU while the status and conditional operations are being modified.

The flow chart for the TxD interrupt procedure is given in Figure 32. For this program five different messages can be transmitted, and they are stored in ROM. It is possible to download the messages into a dedicated RAM buffer; however, the RAM buffer would have to be as large as the largest message. A more efficient way to transmit the messages is to read them from ROM. In this case the address of the first byte of the message would have to be accessible by the transmit interrupt procedure. Since parameters cannot be passed to interrupt procedures, this message pointer is declared PUBLIC in one module and EXTERNAL in the other modules.

To get the transmit interrupt started, the first byte of the message must be written to the transmit buffer. When a section of code decides to transmit a message serially, it loads the global message pointer with the address of the first byte of the message, enables the transmit interrupt, and calls the TxD interrupt procedure. Calling the TxD interrupt procedure writes the first byte to the transmit buffer to initiate transmit interrupts. This can be done by calling PL/M's built-in procedure CAUSE$INTERRUPT.

The transmit interrupt routine checks each byte before it writes it to the transmit buffer. The last character in each message is a 0, so if the character fetched is 0, the transmit interrupt is disabled and the character is ignored.

USING THE LPM WITH THE INTELLEC® MICROCOMPUTER DEVELOPMENT SYSTEM, SERIES II OR SERIES III

A special driver program was written for the MDS to communicate to the LPM. This program, called WRITE, reads a specified file from the disk, expands any TAB characters, and transmits the data through Serial Channel 2 to the LPM. Serial Channel 2 was chosen because $\overline{CTS}$ and $\overline{RTS}$ are brought out to the RS-232 connector. The WRITE program is listed in Appendix B. It was also necessary to modify the boot ROM of the development system so that Serial Channel 2 initializes with $\overline{RTS}$ false and a bit rate of 9600 bps.

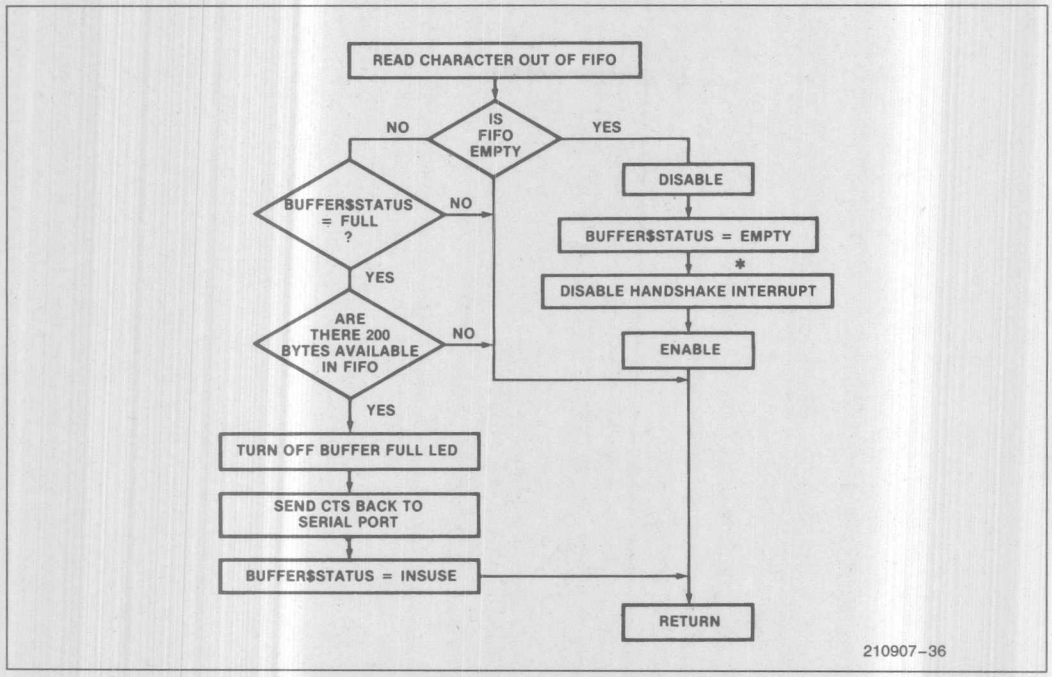


Figure 31. Flow Chart of the BUFF$OUT Procedure

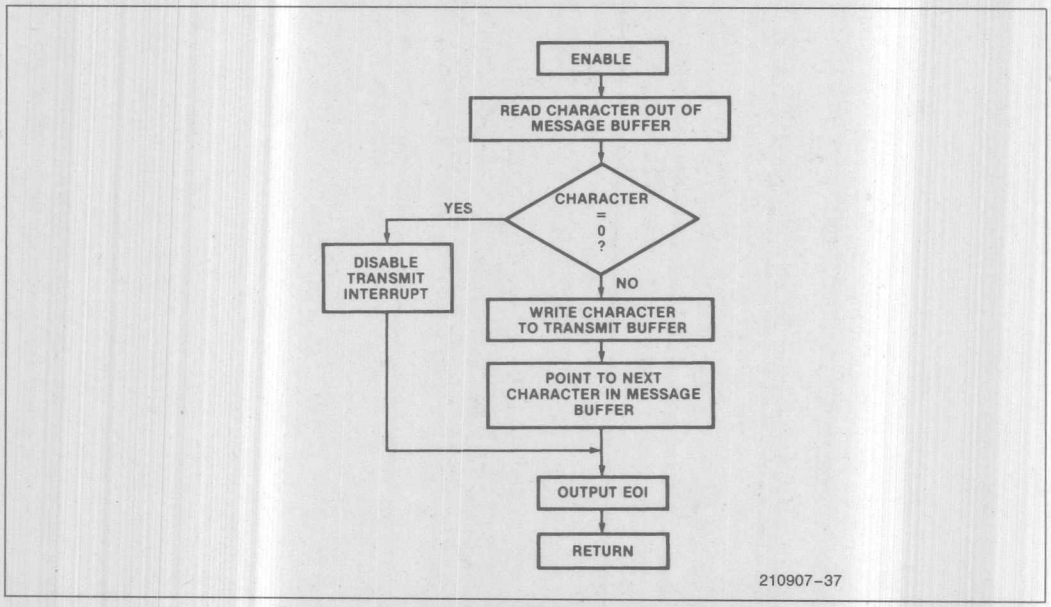


Figure 32. Flow Chart for TxD Interrupt Procedure

Figure 31. Flow Chart of the BUFF$OUT Procedure

Figure 32. Flow Chart for TxD Interrupt Procedure

APPENDIX A
LISTING OF THE LINE PRINTER MULTIPLEXER SOFTWARE

```
PL/M-86 COMPILER    MAINMOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN :F1:MAIN.OBJ
COMPILER INVOKED BY:  PLM86 86  F1:MAIN.SRC


                /*************************************************************************
                 *                                                                       *
                 *              MAIN MODULE FOR THE LINE PRINTER MULTIPLEXER             *
                 *                                                                       *
                 *************************************************************************/

                $DEBUG
  1             MAIN$MOD: DO;

                /*************************************************************************
                * PORT 1 BIT CONFIGURATION                                               *
                *                                                                        *
                *  BUFFER FULL    CTS        ADDRESS     RTS    TWO WIRE HANDSHAKE      *
                *       B7         B6      B5 B4 B3      B2       B1 B0                  *
                *************************************************************************/

  2   1         DECLARE LIT             LITERALLY       'LITERALLY',
                        TRUE            LIT             '0FFH',
                        FALSE           LIT             '0',
                        FOREVER         LIT             'WHILE 1',

                        CMD$1           LIT             '0',    /*8256 REGISTERS*/
                        CMD$2           LIT             '2',
                        CMD$3           LIT             '4',
                        MODE            LIT             '6',
                        PORT$1$CTRL     LIT             '8',
                        SET$INT         LIT             '0AH',
                        INT$EN          LIT             '0AH',
                        RST$INT         LIT             '0CH',
                        INT$ADDR        LIT             '0CH',
                        TX$BUFF         LIT             '0EH',
                        RX$BUFF         LIT             '0EH',
                        PORT$1          LIT             '10H',
                        PORT$2          LIT             '12H',
                        DEBOUNCE$TIMER  LIT             '14H',
                        SCAN$TIMER      LIT             '1AH',
                        RECEIVE$TIMER   LIT             '1CH',
                        STATUS$REG      LIT             '1EH',

                        SCAN$INT        LIT             '40H',
                        DEBOUNCE$INT    LIT             '01H',
                        RECEIVER$INT    LIT             '10H',
                        TIME$OUT$INT    LIT             '08H',
                        TRANSMIT$INT    LIT             '20H',

                        EMPTY           LIT             '0',
                        INUSE           LIT             '1',
                        FULL            LIT             '2',

                        RTS             LIT             '(INPUT(PORT$1) AND 04H)',
```

210907-38

```
                        BEGIN           LABEL   PUBLIC,

                        TEMP            BYTE,
                        SCAN$DELAY      BYTE    PUBLIC,
                        DEBOUNCE$DELAY  BYTE    PUBLIC,
                        RECEIVE$DELAY   BYTE    PUBLIC,
                        PORT$PTR        BYTE    PUBLIC,
                        SERIAL$FORMAT(8)BYTE    PUBLIC, /* PEN EP L1 L0 B3 B2 B1 B0 */

                        MESSAGE$PTR     POINTER EXTERNAL,
                        J               BYTE    EXTERNAL,
                        OK(1)           BYTE    EXTERNAL,
                        BUFFER$STATUS   BYTE    EXTERNAL,

                    /*************************************************************
                     *                EXTERNAL PROCEDURE DECLARATIONS            *
                     *************************************************************/

  3   1             POWER$ON PROCEDURE EXTERNAL,
  4   2             END POWER$ON,

  5   1             LOAD$INT$TABLE PROCEDURE EXTERNAL,
  6   2             END LOAD$INT$TABLE,

                    /*************************************************************
                     *      SET THE BIT RATE AND DATA FORMAT FOR THE SERIAL PORT  *
                     *************************************************************/

  7   1             CONFIGURE PROCEDURE ; /*Initialize bit rate and data format*/
  8   2             TEMP=SERIAL$FORMAT(SHR(PORT$PTR,3));
  9   2             OUTPUT(CMD$1)=((SHL(TEMP,2) AND 0C0H) OR 03H),
 10   2             OUTPUT(CMD$2)=(TEMP OR 30H),
 11   2             END CONFIGURE,

                    /*************************************************************
                     *                INITIALIZE SERIAL RECEIVER                 *
                     *************************************************************/

 12   1             INIT$RECEIVER   PROCEDURE,
 13   2             CALL CONFIGURE,                 /*Initialize 8256 serial port*/
 14   2             RECEIVE$DELAY=TRUE,
 15   2             OUTPUT(CMD$3)=0C0H,             /*Enable serial receiver*/
 16   2             OUTPUT(RECEIVE$TIMER)=70,       /*18 second TIME$OUT*/
 17   2             OUTPUT(SET$INT)=18H,            /*Enable RECEIVER and TIME$OUT interrupts*/
 18   2             IF (BUFFER$STATUS<>FULL)
                        THEN
 19   2               OUTPUT(PORT$1)=(INPUT(PORT$1) AND 0BFH), /*Send CTS TRUE*/

 20   2             DO WHILE RECEIVE$DELAY=TRUE, /* Wait here while receiving serial data */
 21   3             END,

                          /* After 18 seconds of not receiving a character, proceed */

 22   2             OUTPUT(SET$INT)=TRANSMIT$INT,    /* Send the terminating message */
 23   2             J=0,
 24   2             MESSAGE$PTR= @OK(0),
 25   2             CAUSE$INTERRUPT (45H),
```

210907-39

```
PL/M-86 COMPILER    MAINMOD


  26   2      OUTPUT(PORT$1)=(INPUT(PORT$1) OR 40H), /*Send CTS FALSE*/
  27   2      OUTPUT(RST$INT)=18H,                   /*Clear RECEIVER and TIMER Interrupts*/
  28   2      OUTPUT(CMD$3)=40H,                     /*Disable serial receiver*/
  29   2      END INIT$RECEIVER;

              /*****************************************************************
               *                        DEBOUNCE RTS                           *
               *****************************************************************/

  30   1      DEBOUNCE: PROCEDURE;
  31   2      DEBOUNCE$DELAY=TRUE;
  32   2      OUTPUT(DEBOUNCE$TIMER)=10; /* 10 msec debounce time delay */
  33   2      OUTPUT(SET$INT)=DEBOUNCE$INT;
  34   2          DO WHILE DEBOUNCE$DELAY=TRUE;
  35   3          END;
  36   2      IF RTS=0 THEN CALL INIT$RECEIVER;
  38   2      END DEBOUNCE;

              /*****************************************************************
               *                      BEGIN MAIN PROGRAM                       *
               *****************************************************************/


  39   1      BEGIN: CALL POWER$ON;

  40   1      CALL LOAD$INT$TABLE;

  41   1      ENABLE;

  42   1      DO FOREVER;

  43   2      SCAN$DELAY=TRUE;
  44   2      OUTPUT(SCAN$TIMER)=100;     /*Spend 100 msec on each serial port sampling RTS*/
  45   2      OUTPUT(SET$INT)=SCAN$INT;

  46   2          DO WHILE SCAN$DELAY=TRUE;   /*Sample RTS*/
  47   3            IF RTS=0
                       THEN
  48   3              CALL DEBOUNCE;
  49   3          END;


  50   2      TEMP=INPUT(PORT$1);             /*Increment PORT$PTR*/
  51   2      PORT$PTR=TEMP AND 38H;
  52   2      TEMP=TEMP AND (NOT 38H);
  53   2      PORT$PTR=(PORT$PTR+8) AND 38H;

  54   2      OUTPUT(PORT$1)=TEMP OR PORT$PTR; /*Look at next serial port*/
  55   2      END; /*DO FOREVER*/
  56   1      END MAIN$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 011CH    284D
```

```
PL/M-86 COMPILER    MAINMOD


     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 000DH     13D
     MAXIMUM STACK SIZE = 000CH     12D
     159 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

210907-40

```
PL/M-86 COMPILER     INTMOD

SERIES-III PL/M-86 V[illegible] COMPILATION OF MODULE INTMOD
OBJECT MODULE PLACED IN :F1:INT.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:INT.SRC

                /*****************************************************
                 *                                                   *
                 *       INTERRUPT MODULE  CONTAINS ALL INTERRUPT ROUTINES
                 *                         PLUS LOAD INTERRUPT TABLE PROCEDURE
                 *                                                   *
                 *****************************************************/

                $DEBUG
  1             INT$MOD DO.
                $NOLIST

  3   1         DECLARE
                        ESC             LIT             '1BH',
                        SCAN$DELAY      BYTE            EXTERNAL,
                        DEBOUNCE$DELAY  BYTE            EXTERNAL,
                        RECEIVE$DELAY   BYTE            EXTERNAL,
                        MESSAGE$PTR     POINTER         EXTERNAL,
                        J               BYTE            EXTERNAL.

                /*****************************************************
                 *                MESSAGES SENT TO SERIAL PORTS      *
                 *****************************************************/

                        OK (*) BYTE PUBLIC DATA ('TRANSMISSION COMPLETE',0AH,0DH,00),
                        BREAK (*) BYTE PUBLIC DATA ('BREAK DETECT ERROR',0AH,0DH,00),
                        PARITY (*)BYTE PUBLIC DATA ('PARITY ERROR DETECTED',0AH,0DH,00),
                        FRAME (*) BYTE PUBLIC DATA ('FRAMING ERROR DETECTED',0AH,0DH,00),
                        OVER$RUN(*)BYTE PUBLIC DATA('OVER RUN ERROR DETECTED',0AH,0DH,00).

                /*****************************************************
                 *      EXTERNAL PROCEDURES CALLED BY THE INTERRUPT ROUTINES
                 *****************************************************/
  4   1         ERROR: PROCEDURE (STATUS) EXTERNAL.
  5   2         DECLARE STATUS  BYTE;
  6   2         END ERROR;

  7   1         PROGRAM PROCEDURE EXTERNAL.
  8   2         END PROGRAM;

  9   1         BUFF$IN: PROCEDURE (CHAR) EXTERNAL.
 10   2         DECLARE CHAR    BYTE.
 11   2         END BUFF$IN;

 12   1         BUFF$OUT PROCEDURE BYTE EXTERNAL.
 13   2         END BUFF$OUT.

                /*****************************************************
                 *                LOAD THE INTERRUPT TABLE           *
                 *****************************************************/

 14   1         LOAD$INT$TABLE PROCEDURE PUBLIC.
```

210907-41

```
PL/M-86 COMPILER    INTMOD                                                  12/09/82


  15   2      CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
  16   2      CALL SET$INTERRUPT (43H, RECEIVE$TIME),
  17   2      CALL SET$INTERRUPT (44H, RXD),
  18   2      CALL SET$INTERRUPT (45H, TXD),
  19   2      CALL SET$INTERRUPT (46H, SCAN$TIME),
  20   2      CALL SET$INTERRUPT (47H, LP),

  21   2      END LOAD$INT$TABLE;

              /*************************************************************
               *                    INTERRUPT ROUTINES                     *
               *************************************************************/


              /*************************************************************
               *               SET SCAN DELAY FLAG FALSE                   *
               *************************************************************/

  22   1      SCAN$TIME PROCEDURE INTERRUPT 46H;

  23   2      ENABLE;
  24   2      SCAN$DELAY=FALSE;
  25   2      OUTPUT(CMD$3)=88H;              /*Output end for nested mode*/
  26   2      END SCAN$TIME;

              /*************************************************************
               *               SET DEBOUNCE DELAY FLAG FALSE               *
               *************************************************************/

  27   1      DEBOUNCE$TIME: PROCEDURE INTERRUPT 40H;
  28   2      DEBOUNCE$DELAY=FALSE;
  29   2      OUTPUT(CMD$3)=88H;
  30   2      END DEBOUNCE$TIME;

              /*************************************************************
               *               SET RECEIVE DELAY FLAG FALSE                *
               *************************************************************/

  31   1      RECEIVE$TIME PROCEDURE INTERRUPT 43H;
  32   2      ENABLE;
  33   2      RECEIVE$DELAY=FALSE;
  34   2      OUTPUT(CMD$3)=88H;
  35   2      END RECEIVE$TIME;

              /*************************************************************
               *                READ SERIAL RECEIVE BUFFER                 *
               *************************************************************/

  36   1      RXD: PROCEDURE INTERRUPT 44H;

  37   2      DECLARE
                    STATUS          BYTE,
                    CHAR            BYTE;
  38   2      CHAR=INPUT(RX$BUFF),
  39   2      OUTPUT(RECEIVE$TIMER)=70,          /* REINITIALIZE RECEIVE TIME OUT */
  40   2      STATUS=INPUT(STATUS$REG) AND OFH,
```

210907-42

```
PL/M-86 COMPILER    INTMOD

  41   2          IF STATUS<>0
                    THEN
  42   2            CALL ERROR (STATUS);

  43   2          ELSE IF CHAR=ESC
                       THEN
  44   2            CALL PROGRAM;
                  ELSE
  45   2            CALL BUFF$IN (CHAR);
  46   2      OUTPUT(CMD$3)=88H;
  47   2      END RXD;

              /*************************************************************
               *                  SEND A BYTE TO THE LINE PRINTER          *
               *************************************************************/

  48   1      LP: PROCEDURE INTERRUPT 47H;
  49   2      ENABLE;
  50   2      OUTPUT(PORT$2)=BUFF$OUT;
  51   2      OUTPUT(CMD$3)=88H;
  52   2      END LP;

              /*************************************************************
               *                  SEND A BYTE TO THE SERIAL PORTS          *
               *************************************************************/

  53   1      TXD: PROCEDURE INTERRUPT 45H;
  54   2      DECLARE
                  MESSAGE BASED MESSAGE$PTR (1) BYTE,
                  I       BYTE;
  55   2      ENABLE;
  56   2      I=MESSAGE(J);
  57   2          IF I<>0
                     THEN OUTPUT(TX$BUFF)=I;
  59   2          ELSE OUTPUT(RST$INT)=TRANSMIT$INT;
  60   2          J=J+1;
  61   2          OUTPUT(CMD$3)=88H;
  62   2      END TXD;

  63   1      END INT$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 01BDH    445D
     CONSTANT AREA SIZE = 0078H    120D
     VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 0022H     34D
     181 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

210907-43

```
PL/M-86 COMPILER    BUFFMOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE BUFFMOD
OBJECT MODULE PLACED IN  F1 BUFF OBJ
COMPILER INVOKED BY:  PLM86.86  F1 BUFF SRC


                /*************************************************************************
                 *                                                                       *
                 *    BUFFER MODULE:  INSERTS AND REMOVES CHARACTERS FROM FIFO            *
                 *                    REPORTS SERIAL RECEIVE ERRORS AND                  *
                 *                    RE-PROGRAMS SERIAL PORTS                           *
                 *                                                                       *
                 *************************************************************************/

                $DEBUG
  1             BUFF$MOD: DO;
                $NOLIST

  3   1         DECLARE
                        MESSAGE$PTR     POINTER         PUBLIC,
                        J               BYTE            PUBLIC,
                        OK(1)           BYTE            EXTERNAL,
                        BREAK(1)        BYTE            EXTERNAL,
                        PARITY(1)       BYTE            EXTERNAL,
                        FRAME(1)        BYTE            EXTERNAL,
                        OVER$RUN(1)     BYTE            EXTERNAL,
                        SERIAL$FORMAT(1)BYTE            EXTERNAL,
                        PORT$PTR        BYTE            EXTERNAL,

                        FIFO(8192)      BYTE,
                        IN$POINTER      WORD            PUBLIC,
                        OUT$POINTER     WORD            PUBLIC,
                        BUFFER$STATUS   BYTE            PUBLIC;

                /*************************************************************************
                 *                      INSERT CHARACTER INTO FIFO                       *
                 *************************************************************************/

  4   1         BUFF$IN: PROCEDURE (CHAR) PUBLIC;
  5   2         DECLARE
                        CHAR    BYTE;

  6   2         IF ((BUFFER$STATUS=EMPTY) AND ((CHAR=LF) OR (CHAR=CR)))
                    THEN
  7   2                 DO;
  8   3                   OUTPUT(SET$INT)=HANDSHAKE$INT;  /* Enable two-wire handshake interrupt */
  9   3                   BUFFER$STATUS=INUSE;
 10   3                   OUTPUT(PORT$2)=0;               /* Output NULL character to get
                                                                the interrupt started */
 11   3                 END;

 12   2         FIFO(IN$POINTER)=CHAR;    /* Put CHAR into FIFO and increment pointer */
 13   2         IN$POINTER=((INPOINTER+1) AND 1FFFH);

 14   2         IF (((IN$POINTER+4) AND 1FFFH)=OUT$POINTER) /* If the buffer is full, stop reception */
                    THEN
 15   2                 DO;   /* Send CTS FALSE, and light up buffer full LED */
```

210907-44

```
PL/M-86 COMPILER    BUFFMOD

  16   3                OUTPUT(PORT$1)=((INPUT(PORT$1) OR 40H) AND 7FH),
  17   3                BUFFER$STATUS=FULL,
  18   3              END,
  19   2      END BUFF$IN;

             /*************************************************************************
              *                      REMOVE CHARACTER FROM FIFO                       *
              *************************************************************************/

  20   1      BUFF$OUT PROCEDURE BYTE PUBLIC,
  21   2      DECLARE  CHAR    BYTE,
  22   2      CHAR=FIFO(OUT$POINTER),
  23   2      OUT$POINTER=((OUT$POINTER+1) AND 1FFFH),
  24   2      IF OUT$POINTER=IN$POINTER  /* If the buffer is EMPTY disable the output to LP */
                 THEN
  25   2              DO,
  26   3                DISABLE,
  27   3                BUFFER$STATUS=EMPTY,
  28   3                OUTPUT(RST$INT)=HANDSHAKE$INT,
  29   3                ENABLE,
  30   3              END,

             /*  If the buffer is ready to fill up again then send CTS TRUE */

  31   2      ELSE  IF ((BUFFER$STATUS=FULL) AND (((OUT$POINTER-200) AND 1FFFH)=IN$POINTER))
                    THEN
  32   2                  DO; /* Turn off buffer-full LED and turn on CTS */
  33   3                     OUTPUT(PORT$1)=((INPUT(PORT$1) AND 0BFH) OR 80H);
  34   3                     BUFFER$STATUS=INUSE,
  35   3                  END,
             RETURN CHAR,

  37   2      END BUFF$OUT;

             /*************************************************************************
              *                    SEND ERROR MESSAGE TO SERIAL PORT                  *
              *************************************************************************/

  38   1      ERROR PROCEDURE (STATUS) PUBLIC,
  39   2              DECLARE STATUS   BYTE,
                          MESSAGE  BASED  MESSAGE$PTR(1) BYTE,

  40   2               IF (STATUS AND 02H)>0
                          THEN
  41   2                     STATUS=2,
  42   2      ELSE     IF (STATUS AND 04H)>0
                          THEN
  43   2                     STATUS=3,
  44   2      ELSE     IF (STATUS AND 08H)>0
                          THEN
  45   2                     STATUS=4,
  46   2      ELSE     IF (STATUS AND 01H)>0
                          THEN
  47   2                     STATUS=1,
                      DO CASE STATUS,
  49   3
  50   3                    MESSAGE$PTR=@FRAME(0),
```

210907-45

```
PL/M-86 COMPILER    BUFFMOD


  51   3                 MESSAGE$PTR=@OVER$RUN(0);
  52   3                 MESSAGE$PTR=@PARITY(0);
  53   3                 MESSAGE$PTR=@BREAK(0);
  54   3             END;

  55   2             J=1;          /* Point to second character in string */
  56   2             OUTPUT(SET$INT)=TRANSMIT$INT;
  57   2             OUTPUT(TX$BUFF)=MESSAGE(0);
  58   2     END ERROR;

          /*************************************************************
           *                  RELOAD SERIAL PORT CONFIGURE BYTE        *
           *************************************************************/

  59   1     PROGRAM: PROCEDURE PUBLIC;
  60   2             DECLARE TEMP    BYTE,
                          CHAR    BYTE;

  61   2             DO WHILE (INPUT(STATUS$REG) AND 40H)=0; /* Wait for next byte */
  62   3             END;

  63   2             CHAR=INPUT(RX$BUFF);

  64   2             IF CHAR=0       /* If second byte is 0, exit program mode */
                       THEN
  65   2                   DO;
  66   3                       OUTPUT(RECEIVE$TIMER)=70;
  67   3                       CALL BUFF$IN (CHAR);
  68   3                       RETURN;
  69   3                   END;

  70   2             TEMP=(CHAR AND 0FH);

  71   2             DO WHILE (INPUT(STATUS$REG) AND 40H)=0;
  72   3             END;

  73   2             TEMP=(INPUT(RXBUFF) AND 0FH) OR SHL(TEMP,4);

  74   2             SERIAL$FORMAT (SHR(PORT$PTR,3))=TEMP;
  75   2     END PROGRAM;

  76   1     END BUFF$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 01E4H    484D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 200BH   8203D
     MAXIMUM STACK SIZE = 000AH     10D
     189 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

210907-46

```
PL/M-86 COMPILER    PON_MOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE PON_MOD
OBJECT MODULE PLACED IN :F1:PON.OBJ
COMPILER INVOKED BY:  PLM86 86 :F1:PON.SRC

            $DEBUG

            /*****************************************************************************
             *                                                                           *
             *      POWER ON INITIALIZATION OF THE LINE PRINTER MULTIPLEXER              *
             *                                                                           *
             *****************************************************************************/

  1         PON_MOD: DO;

            $NOLIST

  3   1     DECLARE BUFFER$STATUS    BYTE     EXTERNAL,
                    IN$POINTER       WORD     EXTERNAL,
                    OUT$POINTER      WORD     EXTERNAL,
                    PORT$PTR         BYTE     EXTERNAL,
                    SERIAL$FORMAT(8)BYTE      EXTERNAL;

  4   1     POWER$ON: PROCEDURE PUBLIC;

  5   2     DECLARE I                BYTE;

  6   2     DISABLE;

                              /*  INITIALIZE THE MUART */

  7   2     OUTPUT(CMD$1)=01000011B;          /*8086 MODE, FREQ=1KHz, 1 STOP BIT, &
                                                7 BITS/CHARACTER*/
  8   2     OUTPUT(CMD$2)=10110100B;          /*ODD PARITY, SYSTEM CLOCK=1.024 MHz, &
                                                9600 bps*/
  9   2     OUTPUT(CMD$3)=01111111B;          /*CLEAR CMD$3 REGISTER*/
 10   2     OUTPUT(CMD$3)=10110001B;          /*RESET, INTERRUPT ACKNOWLEDGE ENABLE, &
                                                NESTED INTERRUPT MODE*/
 11   2     OUTPUT(MODE)=10000101B;           /*CASCADE TIMERS 35 FOR THE
                                                RECEIVE$TIME$OUT TIMER, BYTE OUTPUT MODE*/

 12   2     OUTPUT(PORT$1$CTRL)=11111000B;    /*PORT 1  RTS=INPUT, THE REST ARE OUTPUTS*/

 13   2     OUTPUT(PORT$1)=11000000B;         /*POINT TO THE FIRST PORT, CTS IS F^' .,
                                                AND BUFFER IS NOT FULL*/

                    /* INITIALIZE FLAGS, VARIABLES, AND ARRAYS */

 14   2     BUFFER$STATUS=EMPTY;
 15   2     IN$POINTER=0;  OUT$POINTER=0;
 17   2     PORT$PTR=0;

 18   2           DO I=0 TO 7;
```

210907-47

```
PL/M-86 COMPILER    PON_MOD


  19   3              SERIAL$FORMAT(I)=10010100B,     /* ON POWER-UP ALL EIGHT SERIAL PORTS
                                                       DEFAULT TO 9600 bps, ODD PARITY, AND
                                                       7 BITS/CHARACTER*/
  20   3              END;
  21   2      END POWER$ON;
  22   1      END PON_MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 0058H     88D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0001H      1D
     MAXIMUM STACK SIZE = 0002H      2D
     98 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

210907-48

APPENDIX B

```
PL/M-80 COMPILER

ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE WRITEMOD
OBJECT MODULE PLACED IN :F1:WRITE.OBJ
COMPILER INVOKED BY:  F2:PLM80 :F1:WRITE.SRC


                $DEBUG
   1            WRITE$MOD: DO;

                /*****************************************************************
                 *                                                               *
                 *   WRITE PROGRAM:  READS A FILE FROM A DISK AND COPIES         *
                 *                   IT TO SERIAL CHANNEL 2 ON THE MDS           *
                 *                                                               *
                 *   SYNTAX OF WRITE:  WRITE  DEVICE:NAME.EXTENSION              *
                 *                                                               *
                 *****************************************************************/

   2    1       DECLARE  LIT             LITERALLY       'LITERALLY',
                         USART$DATA       LIT             '0F6H',
                         USART$STATUS     LIT             '0F7H',
                         RTS              LIT             '20H',
                         TXEN             LIT             '01H',
                         RXE              LIT             '04H',
                         CR               LIT             '0DH',
                         LF               LIT             '0AH',
                         TAB              LIT             '09H',
                         SP               LIT             '20H',
                         ESC              LIT             '1BH',
                         FORM$FEED        LIT             '0CH';

   3    1       DECLARE  AFT$IN          ADDRESS,
                         FILENAME(15)     BYTE,
                         STATUS           ADDRESS,
                         BUFFER(32000)    BYTE,
                         ACTUAL           ADDRESS,
                         CHAR$COUNT       ADDRESS,
                         BYE(42)          BYTE    INITIAL
                ('WROTE ',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,' TO THE LINE PRINTER',0AH,0DH),
                         I                ADDRESS,
                         J                BYTE;

                /*****************************************************************
                 *                  EXTERNAL SYSTEM LIB PROCEDURES               *
                 *****************************************************************/

   4    1       OPEN
                        PROCEDURE (AFTNPTR,FILE,ACCESS,MODE,STATUS) EXTERNAL;
   5    2                DECLARE (AFTNPTR,FILE,ACCESS,MODE,STATUS) ADDRESS;
   6    2       END OPEN;

   7    1       READ
                        PROCEDURE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL;
   8    2                DECLARE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
   9    2       END READ;

  10    1       WRITE;
```

210907-49

```
PL/M-80 COMPILER

                    PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL;
  11   2            DECLARE (AFTN, BUFFER, COUNT, STATUS) ADDRESS;
  12   2      END WRITE;

  13   1      CLOSE:
                    PROCEDURE (AFTN, STATUS) EXTERNAL;
  14   2            DECLARE (AFTN, STATUS) ADDRESS;
  15   2      END CLOSE;

  16   1      ERROR:
                    PROCEDURE (ERRNUM) EXTERNAL;
  17   2            DECLARE (ERRNUM) ADDRESS;
  18   2      END ERROR;

  19   1      EXIT:
                    PROCEDURE EXTERNAL;
  20   2      END EXIT;

                    /*************************************************************
                     *              WAIT UNTIL USART TRANSMITTER IS READY        *
                     *************************************************************/

  21   1      TXRDY:
                    PROCEDURE;
  22   2            DO WHILE ( (INPUT(USART$STATUS) AND 01H) = 0 );
  23   3            END;
  24   2      END TXRDY;

                    /*************************************************************
                     *                    BEGIN MAIN PROGRAM                     *
                     *************************************************************/
  25   1      BEGIN:
                    STATUS=0;

  26   1                CALL READ(1, .FILENAME, 15, .ACTUAL, .STATUS);  /* Read in file and path name */

  27   1                IF STATUS <> 0
                         THEN
  28   1                   GO TO DONE;

  29   1                CALL OPEN( .AFT$IN, .FILENAME, 1, 0, .STATUS); /* Open up the file */

  30   1                IF STATUS <> 0
                         THEN
  31   1                   GO TO DONE;

  32   1      REPEAT:
                    CALL READ(AFT$IN, .BUFFER, 32000, .ACTUAL, .STATUS);

  33   1                IF STATUS <> 0
                         THEN
  34   1                   GO TO DONE;

  35   1                CHAR$COUNT=0;  /* CHAR$COUNT keeps track of the tab columns in each line */

  36   1                OUTPUT(USART$STATUS)= RTS OR TXEN;
```

210907-50

```
PL/M-80 COMPILER

  37   1          IF BUFFER(0)=FORM$FEED   /* If the first character is a form feed
                                              remove it.  Form feeds are inserted at the
                                              end of a file */
                  THEN
  38   1            DO;
  39   2              BUFFER(0)=00H;
  40   2              CHAR$COUNT=-1;
  41   2            END;

  42   1          DO I = 0 TO (ACTUAL - 1);

  43   2            IF  (BUFFER(I)=TAB)  /* Replace TAB characters with the
                                            appropriate number of spaces */
                    THEN
  44   2                DO;
  45   3                   CALL TXRDY;
  46   3                   OUTPUT(USART$DATA)=SP;
  47   3                   CHAR$COUNT=CHAR$COUNT+1;

  48   3                   DO WHILE ((CHAR$COUNT AND 0007H)<>0);
  49   4                      CALL TXRDY;
  50   4                      OUTPUT(USART$DATA)=SP;
  51   4                      CHAR$COUNT=CHAR$COUNT+1;
  52   4                   END;
  53   3                END;
                    ELSE
  54   2                  IF BUFFER(I)=ESC /* If outputting ESC, then output a
                                              0 next so the LPM does not get
                                                 re-programmed */
                      THEN
  55   2                  DO J=0 TO 1;
  56   3                     CALL TXRDY;
  57   3                     OUTPUT(USART$DATA)=0;
  58   3                  END;
                    ELSE /* If the character is not an ESC or TAB then output it */
  59   2                DO;
  60   3                  CALL TXRDY;
  61   3                  OUTPUT(USART$DATA)=BUFFER(I);
  62   3                  IF (BUFFER(I)>1FH AND BUFFER(I)<>7FH)

                        THEN                    /* Only increment CHAR$COUNT
                                                   for printable characters */
  63   3                   CHAR$COUNT=CHAR$COUNT+1;

  64   3                  IF ( (BUFFER(I)=CR) OR (BUFFER(I)=LF) )
                        THEN                    /* Reset CHAR$COUNT for CR or LF */
  65   3                   CHAR$COUNT=0;
  66   3                END;
  67   2          END;

  68   1          IF ACTUAL = 32000 /*If the file is more than 32K, get some more data */
                  THEN
  69   1            GO TO REPEAT;

  70   1          CALL TXRDY;             /* Terminate file with CR, LF, and FF */
  71   1          OUTPUT(USART$DATA)=CR;
  72   1          CALL TXRDY;
```

210907-51

```
PL/M-80 COMPILER

  73   1              OUTPUT(USART$DATA)=LF;
  74   1              CALL TXRDY;
  75   1              OUTPUT(USART$DATA)=FORM$FEED;

  76   1              OUTPUT(USART$STATUS)=RXE OR TXEN; /* Shut off RTS */

  77   1              CALL CLOSE (AFT$IN, STATUS);

  78   1              DO I=0 TO 14;                  /* Output sign off message */
  79   2                 IF FILENAME(I)=CR
                            THEN
  80   2                       GO TO SKIP;
  81   2                 BYE(I+5)=FILENAME(I);
  82   2              END;

  83   1        SKIP: CALL WRITE(0, BYE, 42, STATUS);

  84   1              GO TO NEXT;

  85   1        DONE:
                      CALL ERROR(STATUS);

  86   1        NEXT:
                      CALL EXIT;

  87   1        END WRITE$MOD;


MODULE INFORMATION:

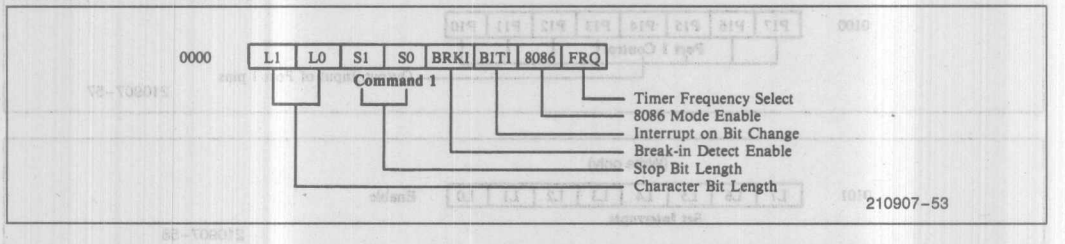
     CODE AREA SIZE     = 0209H    521D
     VARIABLE AREA SIZE = 7D44H  32068D
     MAXIMUM STACK SIZE = 0008H      8D
     191 LINES READ
     0 PROGRAM ERRORS

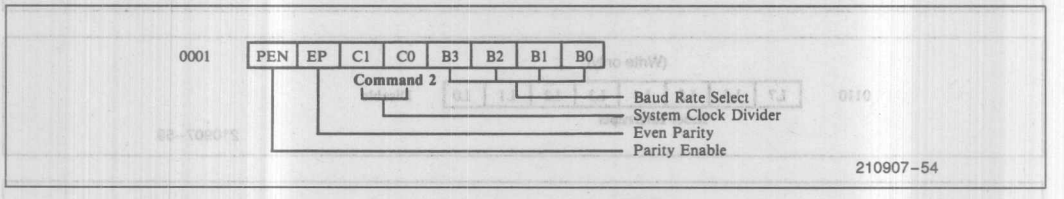
END OF PL/M-80 COMPILATION
```

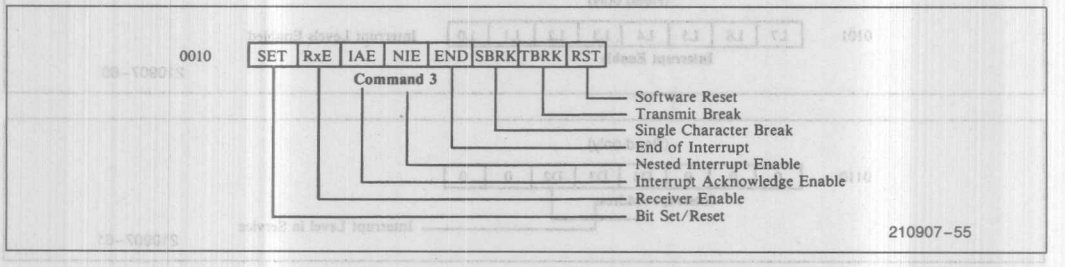
210907-52

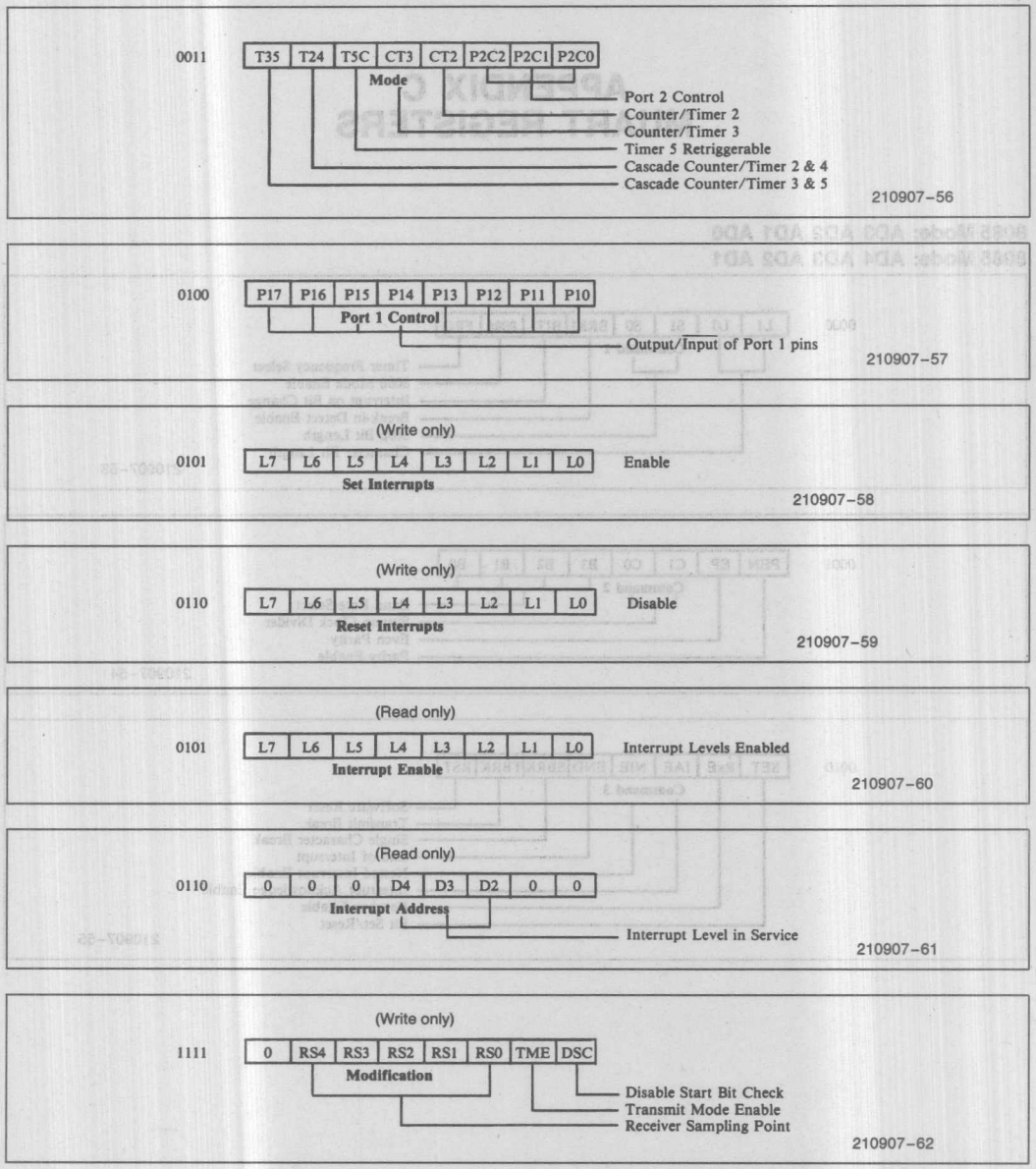
APPENDIX C
MUART REGISTERS

8085 Mode: AD3 AD2 AD1 AD0
8085 Mode: AD4 AD3 AD2 AD1

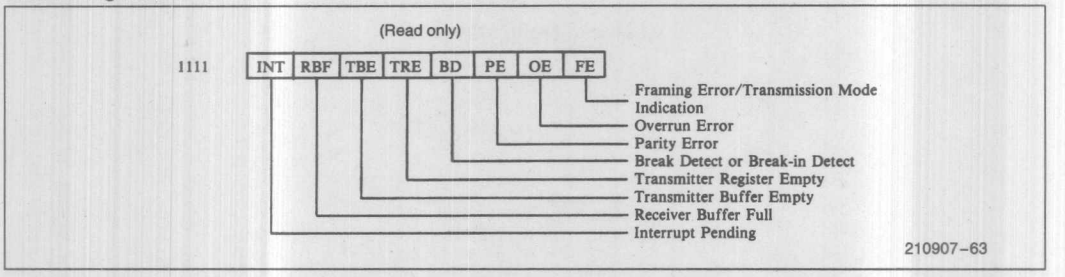




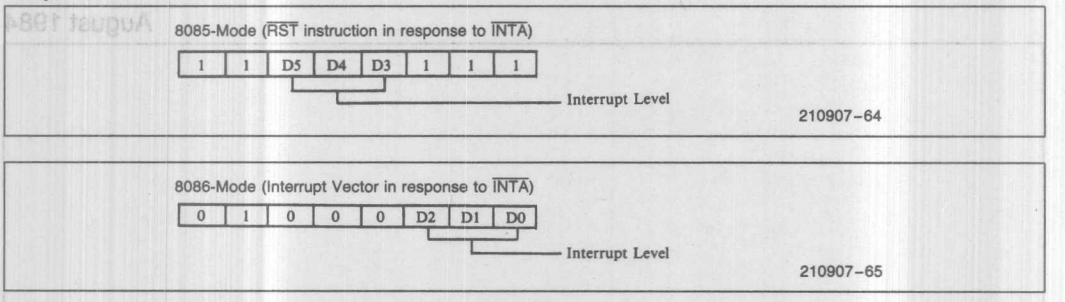



0011
T35 T24 T5C CT3 CT2 P2C2 P2C1 P2C0
Mode
Port 2 Control
Counter/Timer 2
Counter/Timer 3
Timer 5 Retriggerable
Cascade Counter/Timer 2 & 4
Cascade Counter/Timer 3 & 5
210907-56
0100
P17 P16 P15 P14 P13 P12 P11 P10
Port 1 Control
Output/Input of Port 1 pins
210907-57
(Write only)
0101
L7 L6 L5 L4 L3 L2 L1 L0
Enable
Set Interrupts
210907-58
(Write only)
0110
L7 L6 L5 L4 L3 L2 L1 L0
Disable
Reset Interrupts
210907-59
(Read only)
0101
L7 L6 L5 L4 L3 L2 L1 L0
Interrupt Levels Enabled
Interrupt Enable
210907-60
(Read only)
0110
0 0 0 D4 D3 D2 0 0
Interrupt Address
Interrupt Level in Service
210907-61
(Write only)
1111
0 RS4 RS3 RS2 RS1 RS0 TME DSC
Modification
Disable Start Bit Check
Transmit Mode Enable
Receiver Sampling Point
210907-62

Status Register

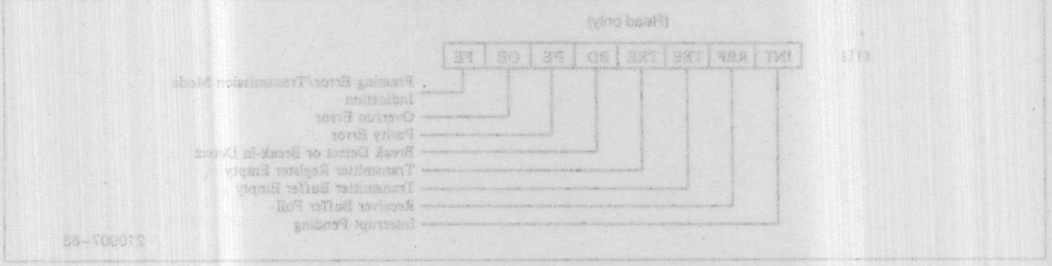


Response to INTA

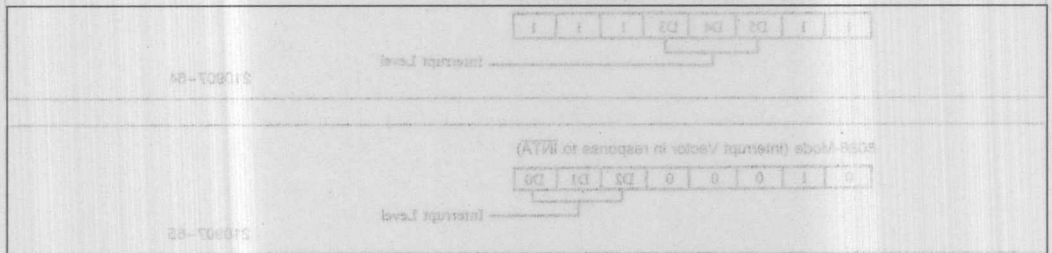


APPLICATION NOTE

AP-183



August 1984



8256AH Multifunction Peripheral Simplifies Microcomputer I/O Design

CHRISTOPHER SCOTT

Order Number: 231125-001

INTRODUCTION

A primary goal of microcomputer system design is to provide the required functionality and flexibility with the fewest number of components. The 8256AH Multi-function Peripheral is designed specifically to meet these conflicting requirements. Four of the most common microcomputer system functions, previously requiring up to four separate MSI or LSI devices, are combined into one LSI device. The 8256AH incorporates a serial asynchronous communication channel, two 8-bit parallel I/O ports, five 8-bit timer/counters and an eight level priority interrupt controller in one 40 pin package. Its flexible design allows it to directly interface to most microprocessors, including Intel's MCS-85, iAPX-86, iAPX-88, iAPX-186 and iAPX-188, and the MCS-48 and MCS-51 family of single-chip microcomputers.

This application note describes using the 8256AH to implement a Data Terminal Equipment (DTE) RS-232C serial asyncrhonous communication link with the control signals necessary to interface to a Bell 103/212A modem. The interface requires a total of nine interface signals. Three of these signals, TxD, RxD and CTS, are provided by the UART section of the 8256AH. The balance of the RS-232C interface signals are implemented using six of the independently programmable parallel PORT 1 lines. In addition, the application design provides an eight bit parallel I/O port with handshaking signals. The on-chip priority interrupt controller enables the RS-232C serial interface and the parallel interface to operate on an interrupt basis. The 8256AH uniquely addresses the complexities of implementing an RS-232C communications interface. By utilizing the built-in hardware and software features of the 8256AH, the design achieves flexibility with simplicity, qualities often exclusive of one another.

Previous solutions required four components to implement the same interface. Figure 1 illustrates the basic system block diagrams for the two solutions. In Figure 1a the 8251A Programmable Communications Interface provides the UART serial communications interface. The 8254 Programmable Interval Timer provides baud rate generation and other timing functions, such as time-out loops, needed for software support of an RS-232C interface. These are especially needed if the RS-232C channel is to operate in an interrupt system environment. The 8255A Programmable Peripheral Interface provides parallel I/O with one port dedicated to the RS-232C control signals. The 8259A Priority Interrupt Controller provides an eight level priority interrupt structure. This represents a total of 120 device pins compared to the single 40 pin 8256AH, and 465 mA current requirement verses a 160 mA current requirement. Figure 1b represents the 8256AH solution incorporating the four functions in one package.

In some data communication applications only three lines - ground, Transmit Data and Receive Data - are used for serial communication. An example is communication between an ASCII terminal or printer and a personal computer. These devices are usually located close to one another and in general do not require the additional control signals of the EIA RS-232C serial communications standard. In other data communications applications, this same equipment requires that the integrity of the serial communications link be constantly monitored. This enables the host system to control the data trransmission at all times, whether it be a host computer or intelligence local to a communications device, such as an ASCII terminal. The need for control and monitoring of the serial line is particularly important when the communications link is over telephone lines using a modem. In a Switched Network, where a number of serial devices share the same communications line, the control signals are crucial to the system's multiplexing the single line.

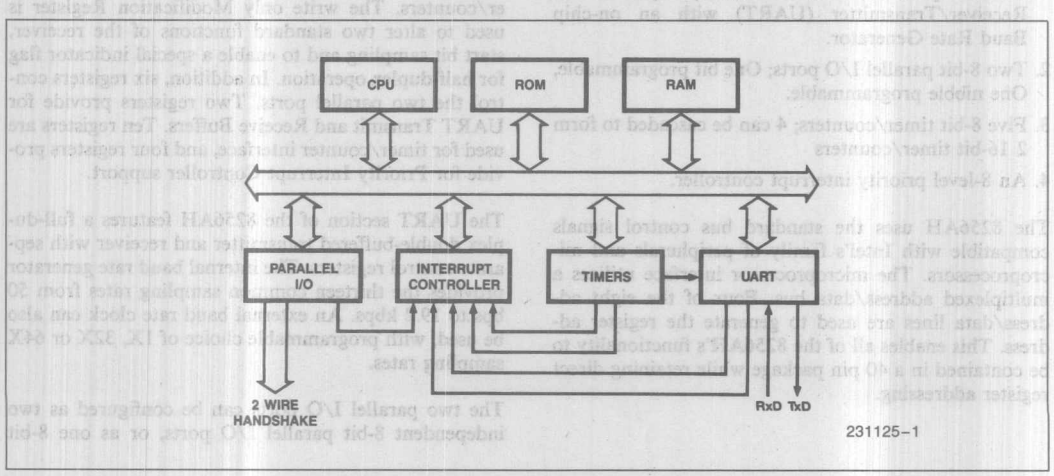


Figure 1a. System Block Diagram Without the 8256AH

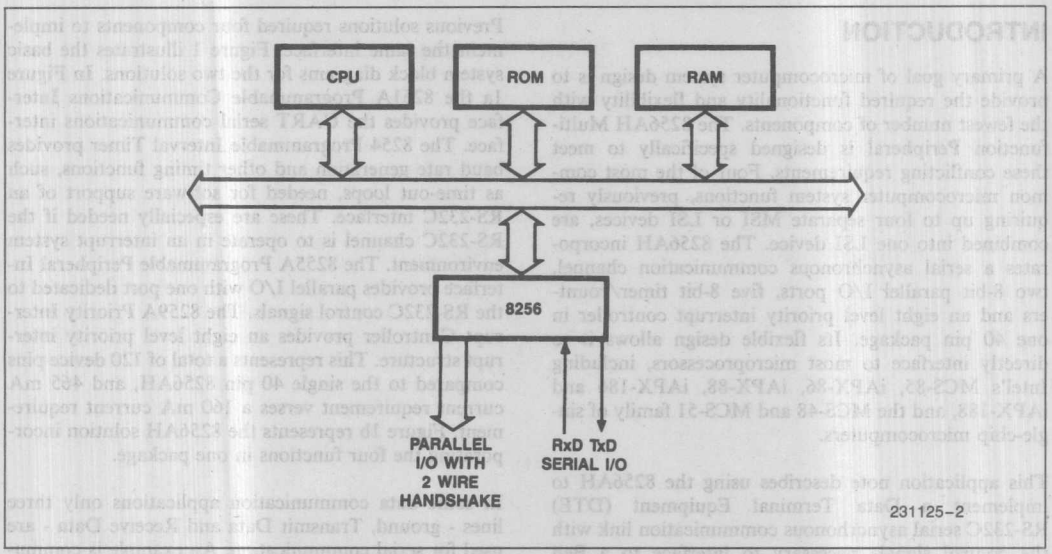


Figure 1b. System Block Diagram With the 8256AH

This Application Note assumes that the reader is familiar with the 8256AH Data Sheet and with the RS-232C communication protocol and terminology. A complete software listing is provided in Appendix A. A complete description and definition of the RS-232C interface standard may be found in the book "Data Communications: A Users Guide" by Kenneth Sherman, Reston Publishing 1981.

DESCRIPTION OF THE 8256AH

The 8256AH combines four commonly used peripheral functions into one device (see Figure 2);

1. A full-duplex, double-buffered serial asynchronous Receiver/Transmitter (UART) with an on-chip Baud Rate Generator.
2. Two 8-bit parallel I/O ports; One bit programmable, One nibble programmable.
3. Five 8-bit timer/counters; 4 can be cascaded to form 2 16-bit timer/counters
4. An 8-level priority interrupt controller.

The 8256AH uses the standard bus control signals compatible with Intel's family of peripherals and microprocessors. The microprocessor interface utilizes a multiplexed address/data bus. Four of the eight address/data lines are used to generate the register address. This enables all of the 8256AH's functionality to be contained in a 40 pin package while retaining direct register addressing.

The sixteen directly addressable internal read/write registers provide control for all of the 8256AH's various functions. Fourteen of the registers are read/write, one, the Status Register, is read only and one, the Modification Register, is write only. Three Command Registers configure the operating environment including the type of CPU, 8 or 16 bit, and system clock frequency. Command Register Three provides bit set-reset capability for control of such functions as End of Interrupt, Nested Interrupts, Interrupt Acknowledge and UART Receive Enable. The Status Register provides all information about the UART's transmitter and receiver, and the state of the interrupt (INT) output pin to the microprocessor. The Mode Register defines the configuration of the two parallel ports and the five timer/counters. The write only Modification Register is used to alter two standard functions of the receiver, start bit sampling and to enable a special indicator flag for half-duplex operation. In addition, six registers control the two parallel ports. Two registers provide for UART Transmit and Receive Buffers. Ten registers are used for timer/counter interface, and four registers provide for Priority Interrupt Controller support.

The UART section of the 8256AH features a full-duplex double-buffered transmitter and receiver with separate control registers. The internal baud rate generator provides the thirteen common sampling rates from 50 bps to 19.2 kbps. An external baud rate clock can also be used, with programmable choice of 1X, 32X or 64X sampling rates.

The two parallel I/O ports can be configured as two independent 8-bit parallel I/O ports, or as one 8-bit

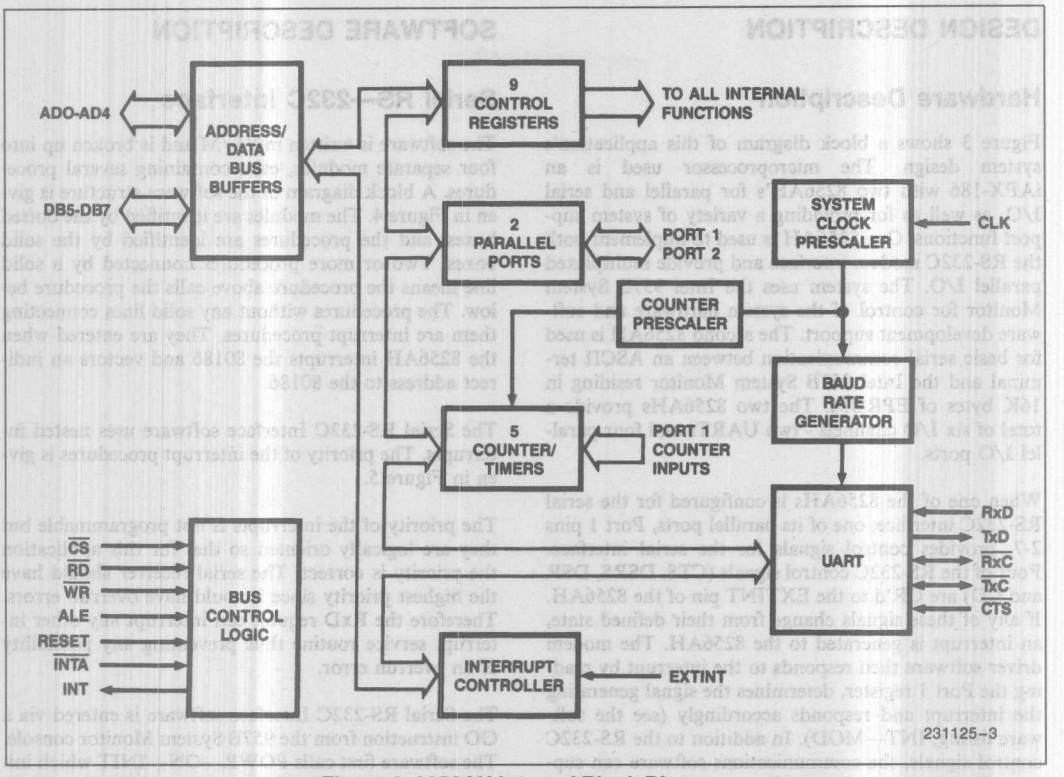


Figure 2. 8256AH Internal Block Diagram

parallel port with ACK/OBF and STB/IBF two wire handshake signals. In the latter configuration, the six remaining I/O lines may be used as either independently programmable I/O lines, or as predefined special function inputs and/or outputs, such as a second external interrupt input or timer/counter inputs.

The five 8-bit programmable timer/counters are binary presettable downcounters. In addition, an independent on-chip Baud Rate Generator is provided for the UART. The clock sources for the timers/counters may be either internal or external - via programmed parallel port pins - depending upon whether they are configured as timers or counters. Four of the timer/counters may be cascaded to form two 16-bit timer/counters. Each of the five timer/counters has its own read/write register.

The eight level priority interrupt controller has twelve possible interrupt sources. Ten of the sources are internal and two are external. One of the external interrupt sources is a fixed pin; EXTINT. The second is one of the parallel Port 1 pins which can be programmed as an external interrupt source. The twelve interrupt sources are internally mapped to the eight interrupt priority levels.

The interrupt controller may be programmed to operate in either a Normal or Nested Interrupt Mode. In Normal Mode any interrupt may interrupt any other interrupt based upon the enable/disable bits in the Interrupt Enable, or Mask, Register. In the Nested Mode only an interrupt of higher piority may interrupt one of lower priority, again based upon the bits in the Enable Register.

The 8256AH interrupt structure supports both 8085 and 8086 interrupt vectoring methods via the INTR and INTA signals. In vectored interrupt operation the 8256AH places the interrupt vector address on the data bus during the INTA sequence. In addition the 8256AH supports non-vectored interrupt interfaces, such as MCS-51 and MCS-48 systems. In non-vectored interrupt applications the host system simply reads the interrupt vector address from the Interrupt Address Register of the 8256AH. Reading the interrupt address register clears the INT pin and acknowledges that the interrupt has been serviced. This is the functional equivalent to an INTA sequence generated by the host processor.

DESIGN DESCRIPTION

Hardware Description

Figure 3 shows a block diagram of this application's system design. The microprocessor used is an iAPX-186 with two 8256AH's for parallel and serial I/O, as well as for providing a variety of system support functions. One 8256AH is used to implement both the RS-232C modem interface and provide multiplexed parallel I/O. The system uses the Intel 957B System Monitor for control of the system hardware and software development support. The second 8256AH is used for basic serial communication between an ASCII terminal and the Intel 957B System Monitor residing in 16K bytes of EPROM. The two 8256AHs provide a total of six I/O channels - two UARTs and four parallel I/O ports.

When one of the 8256AHs is configured for the serial RS-232C interface, one of its parallel ports, Port 1 pins 2-7, provides control signals for the serial interface. Four of the RS-232C control signals (CTS, DSRS, DSR and CD) are OR'd to the EXTINT pin of the 8256AH. If any of these signals change from their defined state, an interrupt is generated to the 8256AH. The modem driver software then responds to the interrupt by reading the Port 1 register, determines the signal generating the interrupt and responds accordingly (see the software listing; INT—MOD). In addition to the RS-232C control signals, the communications software can support all of the standard UART error conditions such as framing errors, underrun, overrun and parity, if parity is enabled.

Parallel I/O With Handshaking

The remaining two Port 1 lines, not used for the RS-232C control signals, provide ACK/OBF and STB/IBF handshaking signals for parallel Port 2. In an environment which utilized the second parallel port, while implementing the above described RS-232C channel, both would operate on an interrupt basis. The interrupt software algorithm depends upon whether the parallel port is configured as input or output, and whether Nested or Normal interrupt mode is programmed. If Nested Interrupt Mode is used, the software flow would default to parallel input or output (as programmed) with Port 2 handshaking the lowest priority interrupt. The serial channel would then interrupt parallel Port 2 transmission whenever the serial channel transmitted or received a character. The RS-232C control signals, OR'd to the External Interrupt (EXTINT) pin, would have the highest interrupt controller priority. The Software Description below describes this in greater detail.

SOFTWARE DESCRIPTION

Serial RS—232C Interface

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 4. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connecting them are interrupt procedures. They are entered when the 8256AH interrupts the 80186 and vectors an indirect address to the 80186.

The Serial RS-232C Interface software uses nested interrupts. The priority of the interrupt procedures is given in Figure 5.

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. The serial receiver should have the highest priority since it could have overrun errors. Therefore the RxD request can interrupt any other interrupt service routine thus preventing any possibility of an overrun error.

The Serial RS-232C Interface software is entered via a GO instruction from the 957B System Monitor console. The software first calls POWR—ON—INIT which initializes the 8256AH. This sets the 8256AH to 8086 Mode with parallel Port 2 in two wire handshake mode using Port 1 pin 0-1 for Port 2 handshaking. The initialization configures six of the Port 1 lines, pins 2-7, for RS-232C handshaking—input or output depending upon the specific signal tied to the pin. Figure 6 illustrates the definition of each Port 1 RS-232C handshaking line and its direction.

Both the Serial RS-232C Interface and the parallel interface with handshaking operate on an interrupt basis. Following initialization the software enters an endless loop and awaits an interrupt from one of three sources; Receive Data (RxD), Transmit Data (TxD) or the parallel interface. In the serial interface idle state, neither transmitting nor receiving data, the software is constantly responding to TxD interrupts; a result of the Transmit Buffer (TBE) and/or Transmit Register (TRE) being continuely empty. When data is received by the RS-232C channel the RxD interrupt, being of higher priority, asserts its interrupt.

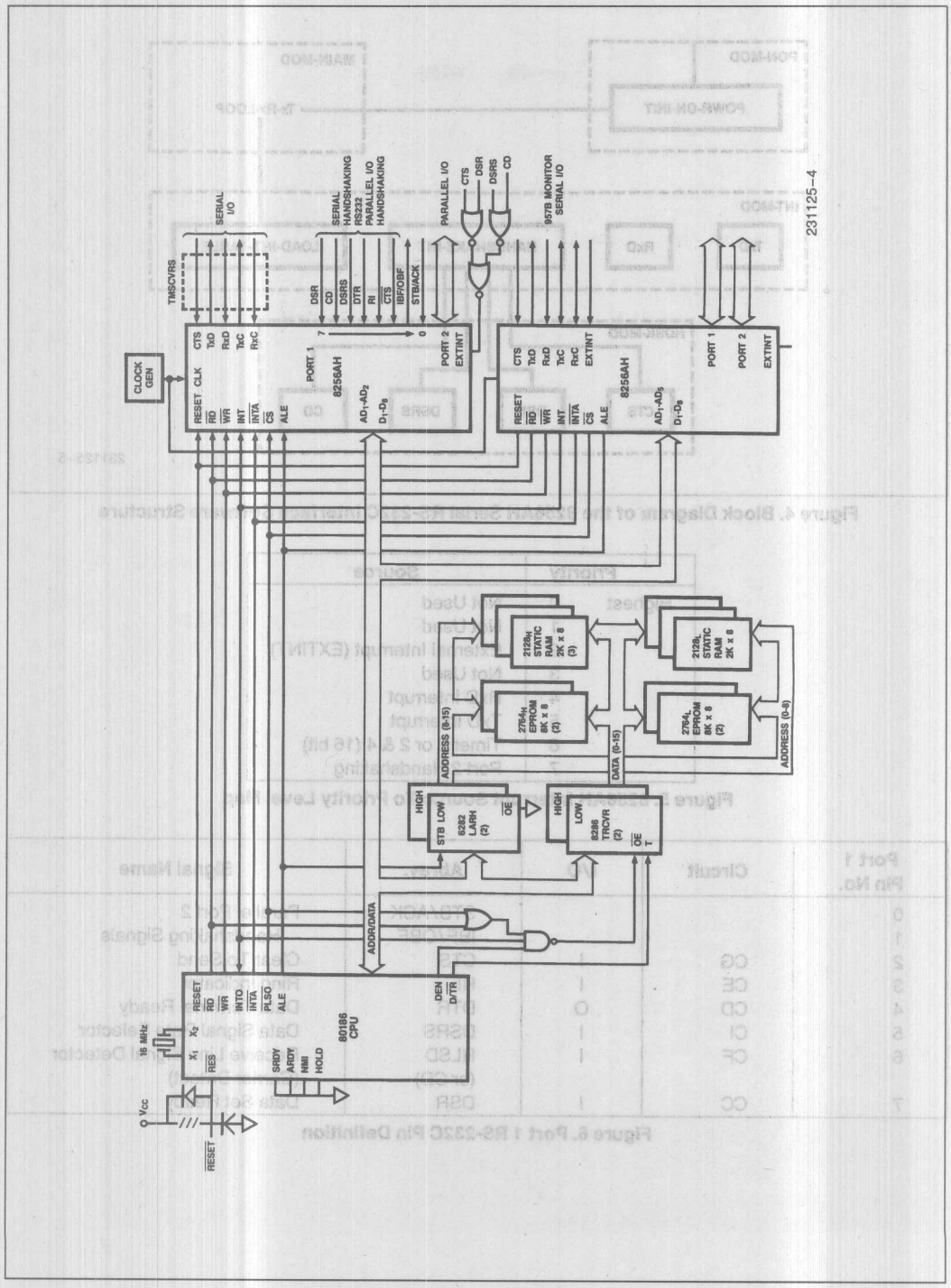


Figure 3. 8256AH / 80186 Schematic

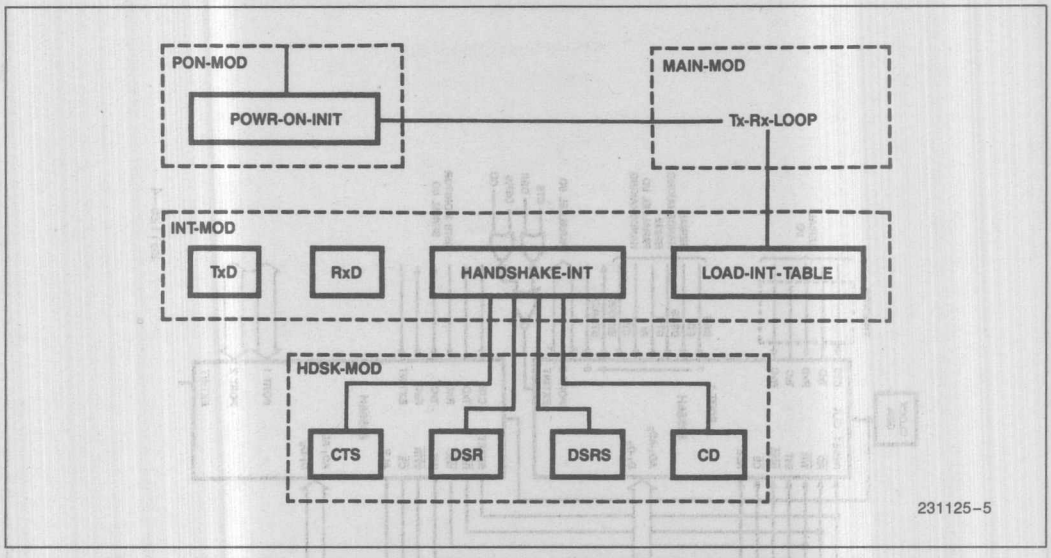


Figure 4. Block Diagram of the 8256AH Serial RS-232C Interface Software Structure

Priority		Source
Highest	0	Not Used
	1	Not Used
	2	External Interrupt (EXTINT)
	3	Not Used
	4	RxD Interrupt
	5	TxD Interrupt
	6	Timer 2 or 2 & 4 (16 bit)
	7	Port 2 Handshaking

Figure 5. 8256AH Interrupt Source To Priority Level Map

Port 1 Pin No.	Circuit	I/O	Abrev.	Signal Name
0			STB/ACK	Parallel Port 2
1			IBF/OBF	Handshaking Signals
2	CG	I	CTS	Clear To Send
3	CE	I	RI	Ring Indicator
4	CD	O	DTR	Data Terminal Ready
5	CI	I	DSRS	Data Signal Rate Selector
6	CF	I	RLSD (or CD)	Receive Line Signal Detector (Carrier Detect)
7	CC	I	DSR	Data Set Ready

Figure 6. Port 1 RS-232C Pin Definition

Although the parallel interface software is not implemented in the software listing of Appendix A, the algorithm for implementing multiplexed parallel and serial I/O is to input or output data on the parallel port during the relatively lengthy time required for serial communication overhead. The algorithm differs slightly during the serial channel idle state when the software responds to repetitive TxD interrupts. In this case the endless loop would detect the idle state repetitive TxD interrupts and disable the TxD interrupt for a short time while the parallel inputs or outputs data. This would require using one of the 8256AH timers to time out repetitive TxD interrupts. The timer used has to be lower in priority than the RxD interrupt to guarantee protection against overrun errors. Timer 2, or 2 and 4 cascaded if longer time delays are desired, provides the proper interrupt level as shown in Figure 5.

Figure 7 shows the Receive Data (RxD) interrupt service routine software flowc art. Since two conditions can generate an RxD Interrupt the Software first reads the Status Register and checks for the Break Detect (DB) bit being set. If the BD bit is clear, no Break condition being present, the data byte is read, stripped to seven bits, for an ASCII character, and sent to the system console via a call to the 957B System Monitor Console Output (CO) routine. Upon return from the 957B monitor call an End Of Interrupt (EOI) is sent to the 8256AH to reset the currently served interrupt level bit in the Interrupt Service Register.

Figure 8 shows the Transmit Data (TxD) interrupt service routine software flowchart. There are three conditions which may cause a TxD Interrupt; TBE, TRE and Break-In Detect. The TxD service routine first reads the Status Register to determine if the interrupt source is the TBE (Transmit Buffer Empty), if not then the interrupt service routine returns to the MAIN—MOD loop. If TBE = 1 (true) then a data byte is read from the 957B System Monitor Console Input (CI) routine. If the data byte is an ASCII character it is written to the 8256AH Transmit Buffer. The software exists via an EOI (End Of Interrupt) command to the 8256AH then returns to the MAIN—MOD Rx—Tx—Loop.

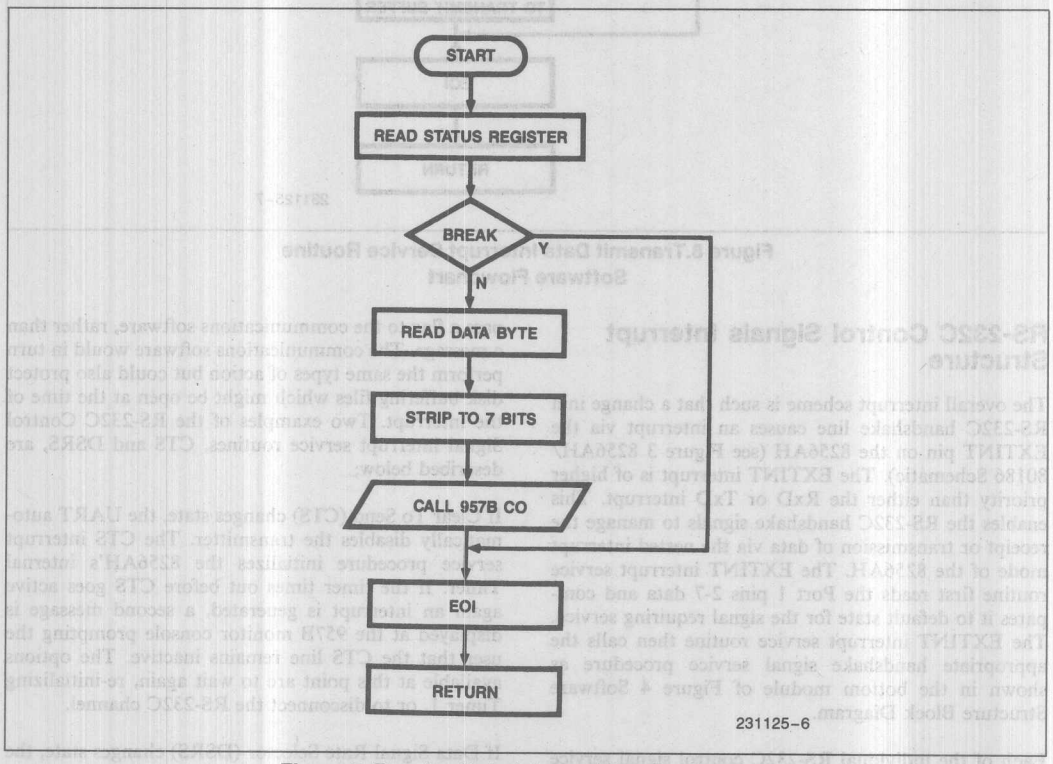


Figure 7. Receive Data Interrupt Service Routine Software Flowchart

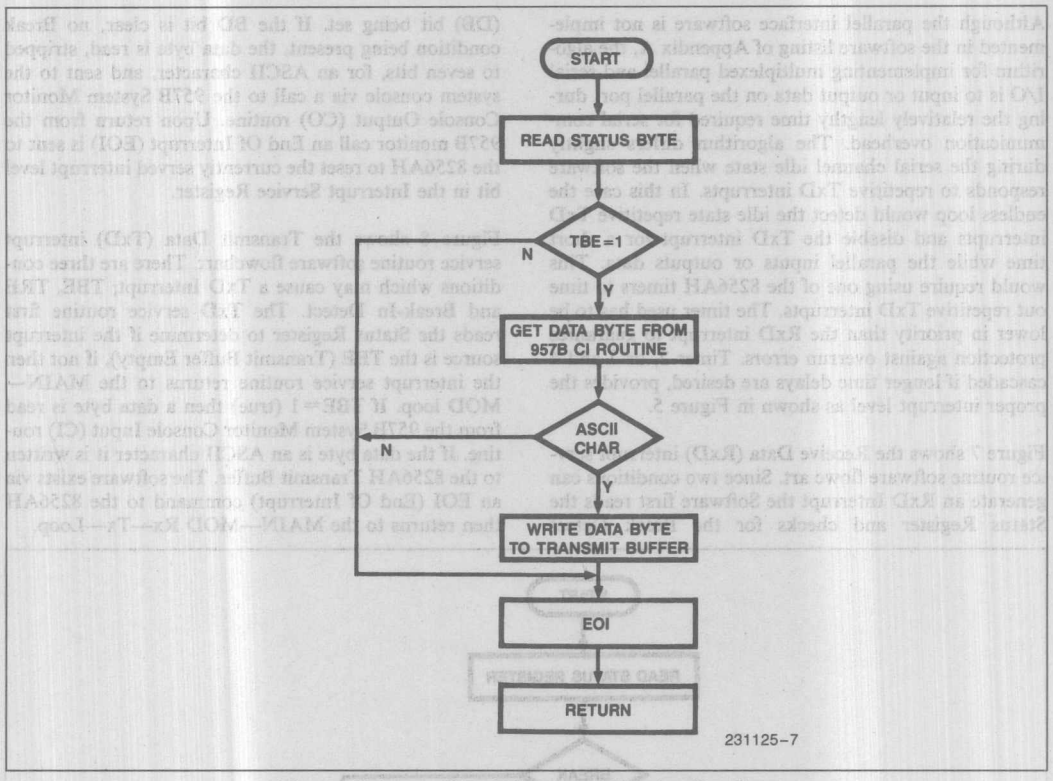


Figure 8.Transmit Data Interrupt Service Routine Software Flowchart

RS-232C Control Signals Interrupt Structure

The overall interrupt scheme is such that a change in a RS-232C handshake line causes an interrupt via the EXTINT pin on the 8256AH (see Figure 3 8256AH/80186 Schematic). The EXTINT interrupt is of higher priority than either the RxD or TxD interrupt. This enables the RS-232C handshake signals to manage the receipt or transmission of data via the nested interrupt mode of the 8256AH. The EXTINT interrupt service routine first reads the Port 1 pins 2-7 data and compares it to default state for the signal requiring service. The EXTINT interrupt service routine then calls the appropriate handshake signal service procedure as shown in the bottom module of Figure 4 Software Structure Block Diagram.

Each of the individual RS-232C control signal service procedures displays a message on the 957B monitor console device indicating the signal requiring a response. The service procedure then either initiates specific predefined actions or prompts the user with options. In a system which utilized file storage, such as a personal computer, the RS-232C software driver could pass a flag to the communications software, rather than a message. The communications software would in turn perform the same types of action but could also protect disk buffering files which might be open at the time of the interrupt. Two examples of the RS-232C Control Signal interrupt service routines, CTS and DSRS, are described below;

If Clear To Send (CTS) changes state, the UART automatically disables the transmitter. The CTS interrupt service procedure initializes the 8256AH's internal Timer. If the timer times out before CTS goes active again an interrupt is generated, a second message is displayed at the 957B monitor console prompting the user that the CTS line remains inactive. The options available at this point are to wait again, re-initializing Timer 1, or to disconnect the RS-232C channel.

If Data Signal Rate Selector (DSRS) changes state, the software prompts the user with a message that the Data Rates of the two RS-232C channels are not the same and the user is given the option of altering the data rate. This application example was interfaced to a 103A/212 Bell modem and as such prompts the user to select between 300 or 1200 bps data rates. In the case of a

non-modem interface the routine could prompt the user for one of the thirteen standard data rates. The software then returns to the TxD/RxD software loop. The balance of the interrupt service procedures for the RS-232C handshaking signals function in a similar manner.

Depending upon the specific system design and software requirements, a variety of enhancements could be added to the system design. These could include interrupt traps that initiate specific corrective options or cascading multiple 8256AHs each with an RS-232C interfaces as described above. An example of an interrupt trap might be auto redial upon time out for lack of Carrier Detect (CD) upon initiating a communications link, or automatic disk file update when a receive buffer approaches overflow.

The ability of the 8256AH to be reprogrammed to meet the changing requirements of a system simplifies the overall system design and multiplies its capabilities. A simple reinitialization sequence could reconfigure the 8256AH as a UART with two parallel ports or utilize any of the various special functions of the parallel Port 1; e.g., an external timer input or an additional external interrupt input, etc. The reinitialization could also configure the 8256AH Multifunction Peripheral for a variety of custom applications.

CONCLUSION

The functional integration of the 8256AH makes it ideal for designs which require maximum flexibility and simplicity of implementation. The implementation of the RS-232C serial channel modem interface and multiplexed parallel I/O described in this application note represent a level of efficiency in peripheral performance and design previously unavailable. The 8256AH Multifunction Peripheral represents a savings of two-thirds the board space and power required by the previous four chip solution, with the added benefit of increased system reliability. The application note demonstrates the ease of implementing the variety of I/O capabilities and system support functions of the 8256AH. The integration of four common microprocessor system functions into one VLSI device enables the designer to devote valuable resources to adding features to enhance the system design, adding performance and flexibility, and reducing the system's overhead.

APPENDIX A

SOFTWARE LISTING

```
PL/M-86 COMPILER    MAINMOD

SERIES-III PL/M-86 V2.3 COMPILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN :F2:56.OBJ
COMPILER INVOKED BY:  PLM86.86 :F2:56

                /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                *                                                               *
                *          8256AH MULTIFUNCTION PERIPHERAL SIMPLIFIES           *
                *              MICROCOMPUTER I/O DESIGN                         *
                *                                                               *
                *                    Intel Corporation                          *
                *                    3065 Bowers Avenue                         *
                *                    Santa Clara,  Ca.  95051                   *
                *                                                               *
                *                                                               *
                *                        Written By     Christopher Scott       *
                *                                                               *
                * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

                $MOD186 DEBUG LARGE
  1             MAINMOD:
                DO;

                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      8256AH Register / Value / Constant  Definitions      */
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  2   1         Declare Lit             Literally       'literally',
                        DCL             lit             'Declare',

                        True            lit             '0ffh',
                        False           lit             '0h',
                        Forever         lit             'while 1',
                        Pcs1            lit             '80h',
                        Cmd1reg         lit             'pcs1 + 0',
                        Cmd2reg         lit             'pcs1 + 2',
                        Cmd3reg         lit             'pcs1 + 4',
                        Modereg         lit             'pcs1 + 6',

                        Port1Ctrl1Reg   lit             'pcs1 + 8',
                        SetIntReg       lit             'pcs1 + 0ah',
                        EnIntReg        lit             'pcs1 + 0ah',
                        RstIntReg       lit             'pcs1 + 0ch',
                        IntAddrReg      lit             'pcs1 + 0ch',
                        TxBuffReg       lit             'pcs1 + 0eh',
                        RxBuffReg       lit             'pcs1 + 0ch',

                        Port1Reg        lit             'pcs1 + 10h',
                        Port2Reg        lit             'pcs1 + 12h',
                        Timer1Reg       lit             'pcs1 + 14h',
                        Timer2Reg       lit             'pcs1 + 1ah',
                        Timer3Reg       lit             'pcs1 + 1ch',
                        StatReg         lit             'pcs1 + 1eh',

                        Intr1           lit             'pcs1 + 40h',
                        Intr2           lit             'cs1 + 01h',
                        Intr3           lit             'pcs1 + 10h',
```

231125-8

```
PL/M-86 COMPILER    MAINMOD

                    Intr4           lit             'pcs1 + 08h',
                    Int_Reset       lit             '88h',
                    SioTxEn         lit             '10h',
                    SioTxRdy        lit             '20h',
                    SioRxRdy        lit             '40h',
                    Break           lit             '04h',
                    DisIntr         lit             '00h',
                    StripTo7fh      lit             '7fh',
                    Port1_Strip     lit             '0fcH',

                    Cmd1            lit             '43h',
                    Cmd2A           lit             '07h',
                    Cmd2B           lit             '09h',
                    Cmd3Clr         lit             '7fh',
                    Cmd3            lit             '0a1h',
                    Mode            lit             '00h',
                    EnRcvr          lit             '0c0h',

                    A               lit             '41h',
                    B               lit             '42h',
                    DSR             lit             '80h',
                    DSR_Flag        lit             '80h',
                    CD              lit             '40h',
                    CD_Flag         lit             '40h',
                    DSRS            lit             '20h',
                    DSRS_Flag       lit             '20h',
                    DTR             lit             '10h',
                    RI              lit             '08h',
                    CTS             lit             '04h',
                    CTS_Flag        lit             '04h',
                    (Status,
                     Hndshk_Pins,
                     J)             Byte,

                    Char            Byte            External,

                    Message_Ptr     Pointer;

                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*          Message Declarations                                 */
  3   1         DCL     CTS_MSG   (*) Byte Public Data ('CTS Disabled. Receive Data stopped. ',
                                                        0AH,0DH,0),
                        DSR_MSG   (*) Byte Public Data ('DSR Disabled. ',0AH,0DH,00),
                        CD_MSG    (*) Byte Public Data ('CD Disabled. ',0AH,0DH,00),
                        DSRS_MSG  (*) Byte Public Data ('Enter Baud Rate:  A. 300   B. 1200
                                                        (A/B)  : ',00),
                        CTS2_MSG  (*) Byte Public Data ('CTS Disabled. Receive Data stopped. ',
                                                        0AH,0DH,00),
                        Break_MSG (*) Byte Public Data ('Break in Receive Data. ',0AH,0DH,00);
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */



                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*          External Procedures:                                 */
```

231125-9

```
PL/M-86 COMPILER    MAINMOD


                    /*                                                    */
                    /*          MCO:   957B Monitor Console Output Routine */
                    /*                                                    */
                    /*          MCI:   957B Monitor Console Input Routine  */
                    /*                                                    */
                    /* - - - - - - - - - - - - - - - - - - - - - - - - - - */

      4   1         MCO: Procedure(Char) External;
      5   2           DCL       Char        Byte;
      6   2         End MCO;

      7   1         MCI: Procedure Byte External;
      8   2         End MCI;

                    /* - - - - - - - - - - - - - - - - - - - - - - - - - - */


                    /* - - - - - - - - - - - - - - - - - - - - - - - - - - */
                    /*      Initialize 8256AH Procedure                   */
      9   1         Init56: Procedure;

     10   2           Disable;
                      /*    Output 8256AH Init Data          */
     11   2           Output(Cmd1Reg)=Cmd1;          /* 8086 mode, freg=1khz, 1 stop bit,
                                                       and 7 bit char */
     12   2           Output(Cmd1Reg)=Cmd2A;         /* odd parity, system clk=1.024mhz,
                                                       and 1200 bps */
     13   2           Output(Cmd1Reg)=Cmd3C1r;       /* clear cmd reg 3 */
     14   2           Output(Cmd1Reg)=Cmd3;          /* reset, itr ack enabled, nested
                                                       intr mode  */
     15   2           Output(Cmd1Reg)=EnRcvr;        /* enable receiver */
     16   2           Output(Cmd1Reg)=Mode;          /* cascade timers 3&5, for the
                                                       receiver$timer$out timer, byte &
                                                       output mode */
     17   2           Call Load_Int_Table;
     18   2           Enable;

     19   2         End Init56;
                    /* - - - - - - - - - - - - - - - - - - - - - - - - - - */


                    /* - - - - - - - - - - - - - - - - - - - - - - - - - - */
                    /*      Procedure:      Load Interrupt Address Vectors */
     20   1         Load_Int_Table: Procedure Public;

     21   2           Call Set$Interrupt(42H,EXTINT);
     22   2           Call Set$Interrupt(44H,Receive_Char);
     23   2           Call Set$Interrupt(45H,Transmit_Char);
     24   2           Call Set$Interrupt(46H,Timer_2_4);

     25   2         End Load_Int_Table;
                    /* - - - - - - - - - - - - - - - - - - - - - - - - - - */


                    /* - - - - - - - - - - - - - - - - - - - - - - - - - - */
                    /*              EXTINT Interrpt Procedure:            */
                    /*                                                    */
                                                                  231125-10
```

```
PL/M-86 COMPILER    MAINMOD


                /*              Service routine reads the Port 1 RS232          */
                /*              handshake signals and sets the message pointer  */
                /*              coresponding to the signal detected.            */
                /*                                                              */
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  26   1        EXTINT: Procedure Interrupt 42H;

  27   2          Enable;
  28   2          HndShk_Pins=Input(Port1Reg) and Port1_Strip;
  29   2          If CTS_Flag = HndShk_Pins and CTS Then
  30   2            Do;
  31   3              Message_Ptr=@CTS_MSG(0);
  32   3              Output(Timer2Reg)=100;
  33   3            End;
  34   2          Else
                    If DSR_Flag = HndShk_Pins and DSR Then
  35   2                Message_Ptr=@DSR_MSG(0);
  36   2              Else
                        If CD_Flag = HndShk_Pins and CD Then
  37   2                    Message_Ptr=@CD_MSG(0);
  38   2                  Else
                            If DSRS_Flag = HndShk_Pins and DSRS Then
  39   2                          DO;
  40   3                            Message_Ptr=@DSRS_MSG(0);
  41   3                            If MCI = A Then
  42   3                              Output(Cmd1Reg)=Cmd2A;      /* odd parity, system clk=1.024mhz,
                                    Else                            and 1200 bps */
  43   3                              If MCI = B Then
  44   3                                Output(Cmd1Reg)=Cmd2B;    /* odd parity, system clk=1.024mhz,
                                                                    and 300 bps */
  45   3                          End;
  46   2          Call Send_Msg;
  47   2          OutPut(RstIntReg)=Int_Reset;

  48   2        End EXTINT;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*       Procedure       Receive a character                   */
  49   1        Receive_Char: Procedure Interrupt 44H;

  50   2          Enable;
  51   2          Status=(Input(StatReg) and SioRxRdy);
  52   2          If Status AND Break Then
  53   2            DO;
  54   3              Message_Ptr=@Break_MSG(0);
  55   3              Call Send_Msg;
  56   3            End;
  57   2          Else
                    Do;
  58   3                Char=Input(RxBuffReg) and StripTo7fh;
  59   3                Call MCO(Char);
  60   3              End;
  61   2          OutPut(RstIntReg)=Int_Reset;

  62   2        End Receive_Char;
```

231125-11

```
PL/M-86 COMPILER    MAINMOD


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      Procedure:      Write character to 8256AH UART                     */
  63   1        Transmit_Char: Procedure Interrupt 45H;

  64   2          Status=(Input(StatReg) and SioRxRdy);
  65   2          If Status and SioRxRdy Then
  66   2            Char=(MCI And StripTo7FH);     /* strip to 7 bits                  */
  67   2            If Char >= 20H And Char <= 7fH Then   /* if char is ASCII output it */
  68   2              Output(TxBuffReg)=Char;
  69   2          OutPut(RstIntReg)=Int_Reset;

  70   2        End Transmit_Char;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      Procedure:      Write character to 856AH UART                      */
  71   1        Timer_2_4: Procedure Interrupt 46H;

  72   2          Message_Ptr=@CTS2_MSG(0);
  73   2          Call Send_Msg;
  74   2          OutPut(RstIntReg)=Int_Reset;

  75   2        End Timer_2_4;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      Message Output Procedure                                          */
  76   1        Send_Msg: Procedure;

  77   2          DCL     Message Based Message_Ptr (1) Byte;

  78   2          J=0;
  79   2          Do While Message(J) <> 0;
  80   3            Char=Message(J);
  81   3            Call MCO(Char);
  82   3            J=J+1;
  83   3          End;
  84   2          Return;

  85   2        End Send_Msg;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      Main Program Body                                                 */

  86   1        Call Init56;
  87   1        Do Forever;
  88   2        End;

                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

231125-12

```
PL/M-86 COMPILER    MAINMOD

  89   1     .End MainMod;


MODULE INFORMATION:

     CODE AREA SIZE     = 0235H    565D
     CONSTANT AREA SIZE = 00BEH    190D
     VARIABLE AREA SIZE = 0007H      7D
     MAXIMUM STACK SIZE = 0034H     52D
     280 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     31KB MEMORY AVAILABLE
     6KB MEMORY USED   (19%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

231125-13

```
PL/M-86 COMPILER   MAINMOD

 ??   1   .End Maintmod;


MODULE INFORMATION:

     CODE AREA SIZE     = 0293H    5BDD
     CONSTANT AREA SIZE = 0088H    190D
     VARIABLE AREA SIZE = 0007H    7D
     MAXIMUM STACK SIZE = 0034H    52D
     256 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     31KB MEMORY AVAILABLE
     8KB MEMORY USED   (17%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

Floppy Disk Controllers

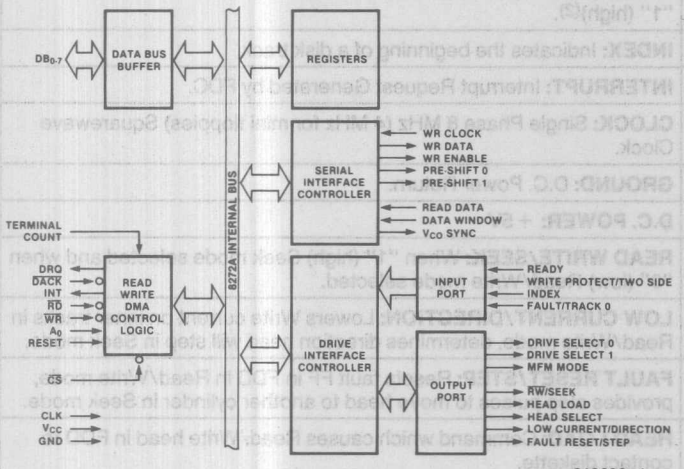
3

3 Floppy Disk Controllers

8272A
SINGLE/DOUBLE DENSITY FLOPPY DISK CONTROLLER

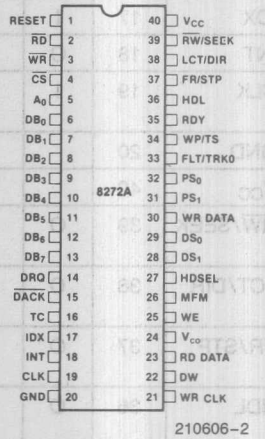
- **IBM Compatible in Both Single and Double Density Recording Formats**
- **Programmable Data Record Lengths: 128, 256, 512, or 1024 Bytes/Sector**
- **Multi-Sector and Multi-Track Transfer Capability**
- **Drives Up to 4 Floppy or Mini-Floppy Disks**
- **Controls 8", 5¼" and 3½" Floppy Disk Drives**
- **Data Transfers in DMA or Non-DMA Mode**
- **Parallel Seek Operations on Up to Four Drives**
- **Compatible with all Intel and Most Other Microprocessors**
- **Single-Phase 8 MHz Clock**
- **Single +5V Power Supply (±10%)**
- **Plastic 40 Pin DIP or 40 Pin CERDIP Packages**

The 8272A is an LSI Floppy Disk Controller (FDC) Chip, which contains the circuitry and control functions for interfacing a processor to 4 Floppy Disk Drives. It is capable of supporting either IBM 3740 single density format (FM), or IBM System 34 Double Density format (MFM) including double sided recording. The 8272A provides control signals which simplify the design of an external phase locked loop and write precompensation circuitry. The FDC simplifies and handles most of the burdens associated with implementing a Floppy Disk Drive Interface. The 8272A is a pin-compatible upgrade to the 8272.



210606-1

Figure 1. 8272A Internal Block Diagram



210606-2

Figure 2. Pin Configuration

November 1986
Order Number: 210606-002

Table 1. Pin Description

Symbol	Pin No.	Type	Connection To	Name and Function
RESET	1	I	μP	**RESET:** Places FDC in idle state. Resets output lines to FDD to "0" (low). Does not clear the last specify command.
$\overline{RD}$	2	I(1)	μP	**READ:** Control signal for transfer of data from FDC to Data Bus, when "0" (low).
$\overline{WR}$	3	I(1)	μP	**WRITE:** Control signal for transfer of data to FDC via Data Bus, when "0" (low).
$\overline{CS}$	4	I	μP	**CHIP SELECT:** IC selected when "0" (low) allowing $\overline{RD}$ and $\overline{WR}$ to be enabled.
A_0	5	I(1)	μP	**DATA/STATUS REGISTER SELECT:** Selects Data Reg (A_0 = 1) or Status Reg (A_0 = 0) contents to be sent to Data Bus.
DB_0–DB_7	6–13	I/O(1)	μP	**DATA BUS:** Bidirectional 8-Bit Data Bus.
DRQ	14	O	DMA	**DATA DMA REQUEST:** DMA Request is being made by FDC when DRQ "1".(3)
$\overline{DACK}$	15	I	DMA	**DMA ACKNOWLEDGE:** DMA cycle is active when "0" (low) and Controller is performing DMA transfer.
TC	16	I	DMA	**TERMINAL COUNT:** Indicates the termination of a DMA transfer when "1" (high)(2).
IDX	17	I	FDD	**INDEX:** Indicates the beginning of a disk track.
INT	18	O	μP	**INTERRUPT:** Interrupt Request Generated by FDC.
CLK	19	I		**CLOCK:** Single Phase 8 MHz (4 MHz for mini floppies) Squarewave Clock.
GND	20			**GROUND:** D.C. Power Return.
V_{CC}	40			**D.C. POWER:** +5V
$\overline{RW}$/SEEK	39	O	FDD	**READ WRITE/SEEK:** When "1" (high) Seek mode selected and when "0" (low) Read/Write mode selected.
LCT/DIR	38	O	FDD	**LOW CURRENT/DIRECTION:** Lowers Write current on inner tracks in Read/Write mode, determines direction head will step in Seek mode.
FR/STP	37	O	FDD	**FAULT RESET/STEP:** Resets fault FF in FDD in Read/Write mode, provides step pulses to move head to another cylinder in Seek mode.
HDL	36	O	FDD	**HEAD LOAD:** Command which causes Read/Write head in FDD to contact diskette.
RDY	35	I	FDD	**READY:** Indicates FDD is ready to send or receive data. Must be tied high (gated by the index pulse) for mini floppies which do not normally have a Ready line.
WP/TS	34	I	FDD	**WRITE PROTECT/TWO-SIDE:** Senses Write Protect status in Read/Write mode, and Two Side Media in Seek mode.
FLT/TRK0	33	I	FDD	**FAULT/TRACK 0:** Senses FDD fault condition in Read/Write mode and Track 0 condition in Seek mode.
PS_1, PS_0	31, 32	O	FDD	**PRECOMPENSATION (PRE-SHIFT):** Write precompensation status during MFM mode. Determines early, late, and normal times.
WR DATA	30	O	FDD	**WRITE DATA:** Serial clock and data bits to FDD.
DS_1, DS_0	28, 29	O	FDD	**DRIVE SELECT:** Selects FDD unit.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Connection To	Name and Function
HDSEL	27	O	FDD	**HEAD SELECT:** Head 1 selected when "1" (high) Head 0 selected when "0" (low).
MFM	26	O	PLL	**MFM MODE:** MFM mode when "1," FM mode when "0".
WE	25	O	FDD	**WRITE ENABLE:** Enables write data into FDD.
VCO	24	O	PLL	**VCO SYNC:** Inhibits VCO in PLL when "0" (low), enables VCO when "1."
RD DATA	23	I	FDD	**READ DATA:** Read data from FDD, containing clock and data bits.
DW	22	I	PLL	**DATA WINDOW:** Generated by PLL, and used to sample data from FDD.
WR CLK	21	I		**WRITE CLOCK:** Write data rate to FDD FM = 500 kHz, MFM = 1 MHz, with a pulse width of 250 ns for both FM and MFM. Must be enabled for all operations, both Read and Write.

NOTES:
1. Disabled when $\overline{CS}$ = 1.
2. TC must be activated to terminate the Execution Phase of any command.
3. DRQ is also an input for certain test modes. It should have a 5 kΩ pull-up resistor to prevent activation.

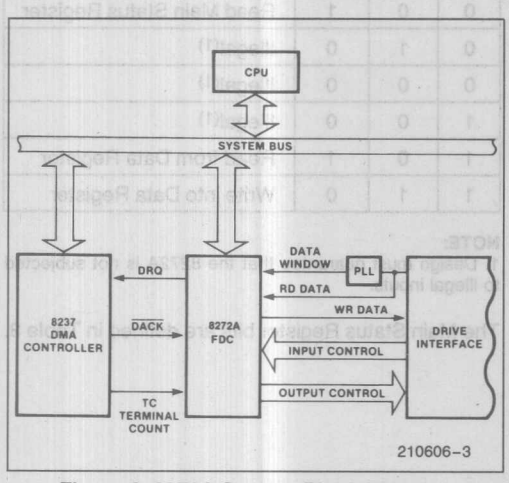


Figure 3. 8272A System Block Diagram

DESCRIPTION

Hand-shaking signals are provided in the 8272A which make DMA operation easy to incorporate with the aid of an external DMA Controller chip, such as the 8237A. The FDC will operate in either DMA or Non-DMA mode. In the Non-DMA mode, the FDC generates interrupts to the processor for every transfer of a data byte between the CPU and the 8272A. In the DMA mode, the processor need only load a command into the FDC and all data transfers occur under control of the 8272A and DMA controller.

There are 15 separate commands which the 8272A will execute. Each of these commands require multiple 8-bit bytes to fully specify the operation which the processor wishes the FDC to perform. The following commands are available.

Read Data	Write Data
Read ID	Format a Track
Read Deleted Data	Write Deleted Data
Read a Track	Seek
Scan Equal	Recalibrate (Restore to Track 0)
Scan High or Equal	Sense Interrupt Status
Scan Low or Equal	Sense Drive Status
Specify	

For more information see the Intel Application Notes AP-116 and AP-121.

FEATURES

Address mark detection circuitry is internal to the FDC which simplifies the phase locked loop and read electronics. The track stepping rate, head load time, and head unload time may be programmed by the user. The 8272A offers many additional features such as multiple sector transfers in both read and write modes with a single command, and full IBM compatibility in both single (FM) and double density (MFM) modes.

8272A ENHANCEMENTS

On the 8272A, after detecting the Index Pulse, the VCO Sync output stays low for a shorter period of time. See Figure 4a.

On the 8272 there can be a problem reading data when Gap 4A is 00 and there is no IAM. This occurs on some older floppy formats. The 8272A cures this problem by adjusting the VCO Sync timing so that it is not low during the data field. See Figure 4b.

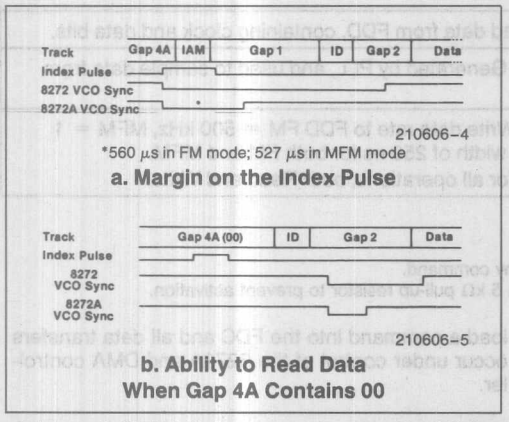


Figure 4. 8272A Enhancements over the 8272

8272A REGISTERS—CPU INTERFACE

The 8272A contains two registers which may be accessed by the main system processor; a Status Register and a Data Register. The 8-bit Main Status Register contains the status information of the FDC, and may be accessed at any time. The 8-bit Data Register (actually consists of several registers in a stack with only one register presented to the data bus at a time), stores data, commands, parameters, and FDD status information. Data bytes are read out of, or written into, the Data Register in order to program or obtain the results after execution of a command. The Status Register may only be read and is used to facilitate the transfer of data between the processor and 8272A.

The relationship between the Status/Down registers and the signals $\overline{RD}$, $\overline{WR}$, and A_0 is shown in Table 2.

Table 2. A_0, $\overline{RD}$, $\overline{WR}$ Decoding for the Selection of Status/Data Register Functions.

A_0	$\overline{RD}$	$\overline{WR}$	Function
0	0	1	Read Main Status Register
0	1	0	Illegal(1)
0	0	0	Illegal(1)
1	0	0	Illegal(1)
1	0	1	Read from Data Register
1	1	0	Write into Data Register

NOTE:
1. Design must guarantee that the 8272A is not subjected to illegal inputs.

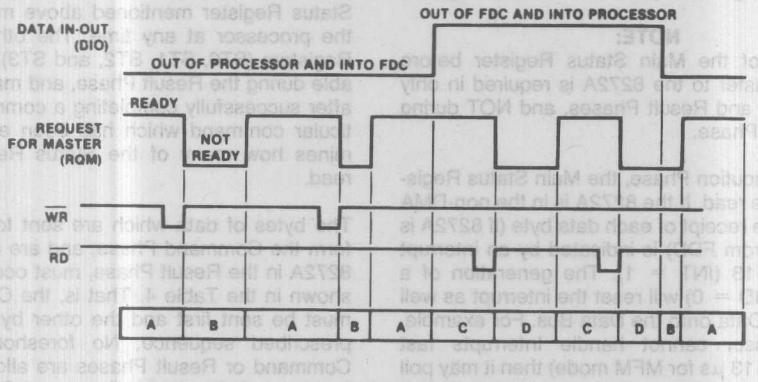
The Main Status Register bits are defined in Table 3.

Table 3. Main Status Register Bit Description

Bit Number	Name	Symbol	Description
D_0	FDD 0 Busy	D_0B	FDD number 0 is in the Seek mode.
D_1	FDD 1 Busy	D_1B	FDD number 1 is in the Seek mode.
D_2	FDD 2 Busy	D_2B	FDD number 2 is in the Seek mode.
D_3	FDD 3 Busy	D_3B	FDD number 3 is in the Seek mode.
D_4	FDC Busy	CB	A read or write command is in process.
D_5	Non-DMA Mode	NDM	The FDC is in the non-DMA mode. This bit is set only during the execution phase in non-DMA mode. Transition to "0" state indicates execution phase has ended.
D_6	Data Input/Output	DIO	Indicates direction of data transfer between FDC and Data Register. If DIO = "1" then transfer is from Data Register to the Processor. If DIO = "0", then transfer is from the Processor to Data Register.
D_7	Request for Master	RQM	Indicates Data Register is ready to send or receive data to or from the Processor. Both bits DIO and RQM should be used to perform the handshaking functions of "ready" and "direction" to the processor.

The DIO and RQM bits in the Status Register indicate when Data is ready and in which direction data will be transferred on the Data Bus.

NOTE:
There is a 12 μs or 24μs RQM flag delay when using an 8 or 4 MHz clock respectively.



NOTES:
A—Data register ready to be written into by processor
B—Data register not ready to be written into by processor
C—Data register ready for next data byte to be read by the processor
D—Data register not ready for next data byte to be read by processor

Figure 5. Status Register Timing

The 8272A is capable of executing 15 different commands. Each command is initiated by a multi-byte transfer from the processor, and the result after execution of the command may also be a multi-byte transfer back to the processor. Because of this multi-byte interchange of information between the 8272A and the processor, it is convenient to consider each command as consisting of three phases:

Command Phase:	The FDC receives all information required to perform a particular operation from the processor.
Execution Phase:	The FDC performs the operation it was instructed to do.
Result Phase:	After completion of the operation, status and other housekeeping information are made available to the processor.

During Command or Result Phases the Main Status Register (described in Table 3) must be read by the processor before each byte of information is written into or read from the Data Register. Bits D6 and D7 in the Main Status Register must be in a 0 and 1 state, respectively, before each byte of the command word may be written into the 8272A. Many of the commands require multiple bytes, and as a result the Main Status Register must be read prior to each byte transfer to the 8272A. On the other hand, during the Result Phase, D6 and D7 in the Main Status Register must both be 1's (D6 = 1 and D7 = 1) before reading each byte from the Data Register.

NOTE:

This reading of the Main Status Register before each byte transfer to the 8272A is required in only the Command and Result Phases, and NOT during the Execution Phase.

During the Execution Phase, the Main Status Register need not be read. If the 8272A is in the non-DMA Mode, then the receipt of each data byte (if 8272A is reading data from FDD) is indicated by an interrupt signal on pin 18 (INT = 1). The generation of a Read signal (RD = 0) will reset the interrupt as well as output the Data onto the Data Bus. For example, if the processor cannot handle Interrupts fast enough (every 13 μs for MFM mode) then it may poll the Main Status Register and then bit D7 (RQM) functions just like the Interrupt signal. If a Write Command is in process, then the WR signal performs the reset to the Interrupt signal.

The 8272A always operates in a multi-sector transfer mode. It continues to transfer data until the TC input is active. In Non-DMA Mode, the system must supply the TC input.

If the 8272A is in the DMA Mode, no Interrupts are generated during the Execution Phase. The 8272A generates DRQ's (DMA Requests) when each byte of data is available. The DMA Controller responds to this request with both a $\overline{DACK}$ = 0 (DMA Acknowledge) and a $\overline{RD}$ = 0 (Read signal). When the DMA Acknowledge signal goes low ($\overline{DACK}$ = 0) then the DMA Request is reset (DRQ = 0). If a Write Command has been programmed then a $\overline{WR}$ signal will appear instead of $\overline{RD}$. After the Execution Phase has been completed (Terminal Count has occurred) then an Interrupt will occur (INT = 1). This signifies the beginning of the Result Phase. When the first byte of data is read during the Result Phase, the Interrupt is automatically reset (INT = 0).

It is important to note that during the Result Phase all bytes shown in the Command Table must be read. The Read Data Command, for example has seven bytes of data in the Result Phase. All seven bytes must be read in order to successfully complete the Read Data Command. The 8272A will not accept a new command until all seven bytes have been read. Other commands may require fewer bytes to be read during the Result Phase.

The 8272A contains five Status Registers. The Main Status Register mentioned above may be read by the processor at any time. The other four Status Registers (ST0, ST1, ST2, and ST3) are only available during the Result Phase, and may be read only after successfully completing a command. The particular command which has been executed determines how many of the Status Registers will be read.

The bytes of data which are sent to the 8272A to form the Command Phase, and are read out of the 8272A in the Result Phase, must occur in the order shown in the Table 4. That is, the Command Code must be sent first and the other bytes sent in the prescribed sequence. No foreshortening of the Command or Result Phases are allowed. After the last byte of data in the Command Phase is sent to the 8272A, the Execution Phase automatically starts. In a similar fashion, when the last byte of

Table 4. 8272A Command Set

Phase	R/W	Data Bus D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID Information
	W				H					Prior to Command
	W				R					Execution
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data Transfer Between the FDD and Main-System
Result	R				ST 0					Status Information
	R				ST 1					After Command
	R				ST 2					Execution
	R				C					
	R				H					Sector ID Information
	R				R					After Command
	R				N					Execution
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID Information
	W				H					Prior to Command
	W				R					Execution
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data Transfer Between the FDD and Main-System
Result	R				ST 0					Status Information
	R				ST 1					After Command
	R				ST 2					Execution
	R				C					
	R				H					Sector ID Information
	R				R					After Command
	R				N					Execution

Table 4. 8272A Command Set (Continued)

Phase	R/W	Data Bus D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W					C				Sector ID Information
	W					H				Prior to Command
	W					R				Execution
	W					N				
	W					EOT				
	W					GPL				
	W					DTL				
Execution										Data Transfer Between the Main-System and FDD
Result	R					ST 0				Status Information
	R					ST 1				After Command
	R					ST 2				Execution
	R					C				
	R					H				Sector ID Information
	R					R				After Command
	R					N				Execution
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W					C				Sector ID Information
	W					H				Prior to Command
	W					R				Execution
	W					N				
	W					EOT				
	W					GPL				
	W					DTL				
Execution										Data Transfer Between the FDD and Main-System
Result	R					ST 0				Status Information
	R					ST 1				After Command
	R					ST 2				Execution
	R					C				
	R					H				Sector ID Information
	R					R				After Command
	R					N				Execution

Table 4. 8272A Command Set (Continued)

Phase	R/W	Data Bus D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID Information
	W				H					Prior to Command
	W				R					Execution
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data Transfer Between the FDD and Main-System. FDC Reads all of Cylinders Contents from Index Hole to EOT
Result	R				ST 0					Status Information
	R				ST 1					After Command
	R				ST 2					Execution
	R				C					
	R				H					Sector ID Information
	R				R					After Command
	R				N					Execution
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Commands
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The First Correct ID Information on the Cylinder is Stored in Data Register
Result	R				ST 0					Status Information
	R				ST 1					After Command
	R				ST 2					Execution
	R				C					
	R				H					Sector ID Information
	R				R					During Execution
	R				N					Phase

Table 4. 8272A Command Set (Continued)

Phase	R/W	Data Bus D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				N					Bytes/Sector
	W				SC					Sectors/Cylinder
	W				GPL					Gap 3
	W				D					Filler Byte
Execution										FDC Formats an Entire Cylinder
Result	R				ST 0					Status Information After Command Execution
	R				ST 1					
	R				ST 2					
	R				C					In This Case, the ID Information has no Meaning
	R				H					
	R				R					
	R				N					

Table 4. 8272A Command Set (Continued)

Phase	R/W	Data Bus D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W		———			C		———		Sector ID Information
	W		———			H		———		Prior to Command
	W		———			R		———		Execution
	W		———			N		———		
	W		———			EOT		———		
	W		———			GPL		———		
	W		———			STP		———		
Execution										Data Compared Between the FDD and Main-System
Result	R		———			ST 0		———		Status Information
	R		———			ST 1		———		After Command
	R		———			ST 2		———		Execution
	R		———			C		———		
	R		———			H		———		Sector ID Information
	R		———			R		———		After Command
	R		———			N		———		Execution
SCAN LOW OR EQUAL										
Command	W	MT	MFM	SK	1	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W		———			C		———		Sector ID Information
	W		———			H		———		Prior to Command
	W		———			R		———		Execution
	W		———			N		———		
	W		———			EOT		———		
	W		———			GPL		———		
	W		———			STP		———		
Execution										Data Compared Between the FDD and Main-System
Result	R		———			ST 0		———		Status Information
	R		———			ST 1		———		After Command
	R		———			ST 2		———		Execution
	R		———			C		———		
	R		———			H		———		Sector ID Information
	R		———			R		———		After Command
	R		———			N		———		Execution

Table 4. 8272A Command Set (Continued)

Phase	R/W	Data Bus D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
SCAN HIGH OR EQUAL										
Command	W	MT	MFM	SK	1	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID Information
	W				H					Prior to Command
	W				R					Execution
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data Compared Between the FDD and Main-System
Result	R				ST 0					Status Information
	R				ST 1					After Command
	R				ST 2					Execution
	R				C					
	R				H					Sector ID Information
	R				R					After Command
	R				N					Execution
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head Retracted to Track 0
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R				ST 0					Status Information at
	R				PCN					the End of Each Seek Operation About the FDC

Table 4. 8272A Command Set (Continued)

Phase	R/W	Data Bus D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	—	SRT	——>		<——		HUT	—	
	W	—	HLT	——>					ND	
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Result	R	——	——	——	ST 3	——	——	——	——	Status Information about FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	——	——	——	NCN	——	——	——	——	
Execution										Head is Positioned Over Proper Cylinder on Diskette
INVALID										
Command	W	——	——	——	Invalid Codes	——	——	——	——	Invalid Command Codes (NoOp—FDC Goes Into Standby State)
Result	R	——	——	——	ST 0	——	——	——	——	ST 0 = 80 (16)

Table 5. Command Mnemonics

Symbol	Name	Description
A_0	Address Line 0	A_0 controls selection of Main Status Register (A_0 = 0) or Data Register (A_0 = 1).
C	Cylinder Number	C stands for the current selected Cylinder track number 0 through 76 of the medium.
D	Data	D stands for the data pattern which is going to be written into a Sector.
D_7–D_0	Data Bus	8-bit Data bus where D_7 is the most significant bit, and D_0 is the least significant bit.
DS0, DS1	Drive Select	DS stands for a selected drive number 0 or 1.
DTL	Data Length	When N is defined as 00, DTL stands for the data length which users are going to read out or write into the Sector.
EOT	End of Track	EOT stands for the final Sector number of a Cylinder.
GPL	Gap Length	GPL stands for the length of Gap 3 (spacing between Sectors excluding VCO Sync Field).
H	Head Address	H stands for head number 0 or 1, as specified in ID field.
HDS	Head Select	HDS stands for a selected head number 0 or 1 (H = HDS in all command words).
HLT	Head Load Time	HLT stands for the head load time in the FDD (2 to 254 ms in 2 ms increments).
HUT	Head Unload Time	HUT stands for the head unload time after a read or write operation has occurred (16 to 240 ms in 16 ms increments).
MFM	FM or MFM Mode	If MF is low, FM mode is selected and if it is high, MFM mode is selected.
MT	Multi-Track	If MT is high, a multi-track operation is to be peformed (a cylinder under both HD0 and HD1 will be read or written).
N	Number	N stands for the number of data bytes written in a Sector.
NCN	New Cylinder Number	NCN stands for a new Cylinder number, which is going to be reached as a result of the Seek operation. Desired position of Head.
ND	Non-DMA Mode	ND stands for operation in the Non-DMA Mode.
PCN	Present Cylinder Number	PCN stands for the Cylinder number at the completion of SENSE INTERRUPT STATUS Command. Position of Head at present time.
R	Record	R stands for the Sector number, which will be read or written.
R/W	Read/Write	R/W stands for either Read (R) or Write (W) signal.
SC	Sector	SC indicates the number of Sectors per Cylinder.
SK	Skip	SK stands for Skip Deleted Data Address Mark.
SRT	Step Rate Time	SRT stands for the Stepping Rate for the FDD (1 to 16 ms in 1 ms increments). The same Stepping Rate applies to all drives (F = 1 ms, E = 2 ms, etc.).

Table 5. Command Mnemonics (Continued)

Symbol	Name	Description
ST 0 ST 1 ST 2 ST 3	Status 0 Status 1 Status 2 Status 3	ST 0–3 stand for one of four registers which store the status information after a command has been executed. This information is available during the result phase after command execution. These registers should not be confused with the main status register (selected by $A_0 = 0$). ST 0–3 may be read only after a command has been executed and contain information relevant to that particular command.
STP		During a Scan operation, if STP = 1, the data in contiguous sectors is compared byte by byte with data sent from the processor (or DMA), and if STP = 2, then alternate sectors are read and compared.

data is read out in the Result Phase, the command is automatically ended and the 8272A is ready for a new command. A command may be aborted by simply sending a Terminal Count signal to pin 16 (TC=1). This is a convenient means of ensuring that the processor may always get the 8272A's attention even if the disk system hangs up in an abnormal manner.

POLLING FEATURE OF THE 8272A

After power-up RESET, the Drive Select Lines DS0 and DS1 will automatically go into a polling mode. In between commands (and between step pulses in the SEEK command) the 8272A polls all four FDDs looking for a change in the Ready line from any of the drives. If the Ready line changes state (usually due to a door opening or closing) then the 8272A will generate an interrupt. When Status Register 0 (ST0) is read (after Sense Interrupt Status is issued), Not Ready (NR) will be indicated. The polling of the Ready line by the 8272A occurs continuously between instructions, thus notifying the processor which drives are on or off line. Approximate scan timing is shown in Table 6.

Table 6. Scan Timing

DS1	DS0	Approximate Scan Timing
0	0	220 μs
0	1	220 μs
1	0	220 μs
1	1	440 μs

COMMAND DESCRIPTIONS

During the Command Phase, the Main Status Register must be polled by the CPU before each byte is written into the Data Register. The DIO (DB6) and RQM (BD7) bits in the Main Status Register must be in the "0" and "1" states respectively, before each byte of the command may be written into the 8272A. The beginning of the execution phase for any of these commands will cause DIO and RQM to switch to "1" and "0" states respectively.

READ DATA

A set of nine (9) byte words are required to place the FDC into the Read Data Mode. After the Read Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Address Marks and ID fields. When the current sector number ("R") stored in the ID Register (IDR) compares with the sector number read off the diskette, then the FDC outputs data (from the data field) byte-by-byte to the main system via the data bus.

After completion of the read operation from the current sector, the Sector Number is incremented by one, and the data from the next sector is read and output on the data bus. This continuous function is called a "Multi-Sector Read Operation." The Read Data Command must be terminated by the receipt of a Terminal Count signal. Upon receipt of this signal, the FDC stops outputting data to the processor, but will continue to read data from the current sector, check CRC (Cyclic Redundancy Count) bytes, and then at the end of the sector terminate the Read Data Command.

The amount of data which can be handled with a single command to the FDC depends upon MT (multi-track), MFM (MFM/FM), and N (Number of Bytes/Sector). Table 7 on the next page shows the Transfer Capacity.

Table 7. Transfer Capacity

Multi-Track MT	MFM/FM MFM	Bytes/Sector N	Maximum Transfer Capacity (Bytes/Sector)(Number of Sectors)	Final Sector Read from Diskette
0	0	00	(128) (26) = 3,328	26 at Side 0
0	1	01	(256) (26) = 6,656	or 26 at Side 1
1	0	00	(128) (52) = 6,656	26 at Side 1
1	1	01	(256) (52) = 13,312	
0	0	01	(256) (15) = 3,840	15 at Side 0
0	1	02	(512) (15) = 7,680	or 15 at Side 1
1	0	01	(256) (30) = 7,680	15 at Side 1
1	1	02	(512) (30) = 15,360	
0	0	02	(512) (8) = 4,096	8 at Side 0
0	1	03	(1024) (8) = 8,192	or 8 at Side 1
1	0	02	(512) (16) = 8,192	8 at Side 1
1	1	03	(1024) (16) = 16,384	

The "multi-track" function (MT) allows the FDC to read data from both sides of the diskette. For a particular cylinder, data will be transferred starting at Sector 1, Side 0 and completing at Sector L, Side 1 (Sector L = last sector on the side).

NOTE:
This function pertains to only one cylinder (the same track) on each side of the diskette.

When N = 0, then DTL defines the data length which the FDC must treat as a sector. If DTL is smaller than the actual data length in a Sector, the data beyond DTL in the Sector is not sent to the Data Bus. The FDC reads (internally) the complete Sector performing the CRC check, and depending upon the manner of command termination, may perform a Multi-Sector Read Operation. When N is non-zero, then DTL has no meaning and should be set to 0FFH.

At the completion of the Read Data Command, the head is not unloaded until after Head Unload Time Interval (specified in the Specify Command) has elapsed. If the processor issues another command before the head unloads then the head settling time may be saved between subsequent reads. This time out is particularly valuable when a diskette is copied from one drive to another.

If the FDC detects the Index Hole twice without finding the right sector, (indicated in "R"), then the FDC sets the ND (No Data) flag in Status Register 1 to a 1 (high), and terminates the Read Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

After reading the ID and Data Fields in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in ID field), the FDC sets the DE (Data Error) flag in Status Register 1 to a 1 (high), and if a CRC error occurs in the Data Field the FDC also sets the DD (Data Error in Data Field) flag in Status Register 2 to a 1 (high), and terminates the Read Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

If the FDC reads a Deleted Data Address Mark off the diskette, and the SK bit (bit D5 in the first Command Word) is not set (SK = 0), then the FDC sets the CM (Control Mark) flag in Status Register 2 to a 1 (high), and terminates the Read Data Command, after reading all the data in the Sector. If SK = 1, the FDC skips the sector with the Deleted Data Address Mark and reads the next sector.

During disk data transfers between the FDC and the processor, via the data bus, the FDC must be serviced by the processor every 27 μs in the FM Mode, and every 13 μs in the MFM Mode, or the FDC sets the OR (Over Run) flag in Status Register 1 to a 1 (high), and terminates the Read Data Command.

If the processor terminates a read (or write) operation in the FDC, then the ID Information in the Result Phase is dependent upon the state of the MT bit and EOT byte. Table 5 shows the values for C, H, R, and N, when the processor terminates the Command.

Table 8. ID Information When Processor Terminates Command

MT	EOT	Final Sector Transferred to Processor	ID Information at Result Phase			
			C	H	R	N
0	1A	Sector 1 to 25 at Side 0	NC	NC	R + 1	NC
	0F	Sector 1 to 14 at Side 0				
	08	Sector 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	C + 1	NC	R = 01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sector 1 to 25 at Side 1	NC	NC	R + 1	NC
	0F	Sector 1 to 14 at Side 1				
	08	Sector 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C + 1	NC	R = 01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				
1	1A	Sector 1 to 25 at Side 0	NC	NC	R + 1	NC
	0F	Sector 1 to 14 at Side 0				
	08	Sector 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	NC	LSB	R = 01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sector 1 to 25 at Side 1	NC	NC	R + 1	NC
	0F	Sector 1 to 14 at Side 1				
	08	Sector 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C + 1	LSB	R = 01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				

NOTES:
1. NC (No Change): The same value as the one at the beginning of command execution.
2. LSB (Least Significant Bit): The least significant bit of H is complemented.

WRITE DATA

A set of nine (9) bytes are required to set the FDC into the Write Data mode. After the Write Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Fields. When the current sector number ("R"), stored in the ID Register (IDR) compares with the sector number read off the diskette, then the FDC takes data from the processor byte-by-byte via the data bus, and outputs it to the FDD.

After writing data into the current sector, the Sector Number stored in "R" is incremented by one, and the next data field is written into. The FDC continues this "Multi-Sector Write Operation" until the issuance of a Terminal Count signal. If a Terminal Count signal is sent to the FDC it continues writing into the current sector to complete the data field. If the Terminal Count signal is received while a data field is being written then the remainder of the data field is filled with 00 (zeros).

The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (incorrect CRC) in one of the ID Fields, it sets the DE (Data Error) flag of Status Register 1 to a 1 (high), and terminates the Write Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

The Write Command operates in much the same manner as the Read Command. The following items

are the same; refer to the Read Data Command for details:

- Transfer Capacity
- EN (End of Cylinder) Flag
- ND (No Data) Flag
- Head Unload Time Interval
- ID Information when the processor terminates command (see Table 2)
- Definition of DTL when N = 0 and when N ≠ 0

In the Write Data mode, data transfers between the processor and FDC must occur every 31 μs in the FM mode, and every 15 μs in the MFM mode. If the time interval between data transfers is longer than this then the FDC sets the OR (Over Run) flag in Status Register 1 to a 1 (high), and terminates the Write Data Command.

For mini-floppies, multiple track writes are usually not permitted. This is because of the turn-off time of the erase head coils—the head switches tracks before the erase head turns off. Therefore the system should typically wait 1.3 ms before attempting to step or change sides.

WRITE DELETED DATA

This command is the same as the Write Data Command except a Deleted Data Address Mark is written at the beginning of the Data Field instead of the normal Data Address Mark.

READ DELETED DATA

This command is the same as the Read Data Command except that when the FDC detects a Data Address Mark at the beginning of a Data Field (and SK = 0 (low)), it will read all the data in the sector and set the CM flag in Status Register 2 to a 1 (high), and then terminate the command. If SK = 1, then the FDC skips the sector with the Data Address Mark and reads the next sector.

READ A TRACK

This command is similar to READ DATA Command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering the INDEX HOLE, the FDC starts reading all data fields on the track as continuous blocks of data. If the FDC finds an error in the ID or DATA CRC check bytes, it continues to read data from the track. The FDC compares the ID information read from each sector with the value stored in the IDR, and sets the ND flag of Status Register 1 to a 1 (high) if there is no comparison. Multi-track or skip operations are not allowed with this command.

This command terminates when EOT number of sectors have been read. If the FDC does not find an ID Address Mark on the diskette after it encounters the INDEX HOLE for the second time, then it sets the MA (missing address mark) flag in Status Register 1 to a 1 (high), and terminates the command. (Status Register 0 has bits 7 and 6 set to 0 and 1 respectively.)

READ ID

The READ ID Command is used to give the present position of the recording head. The FDC stores the values from the first ID Field it is able to read. If no proper ID Address Mark is found on the diskette, before the INDEX HOLE is encountered for the second time then the MA (Missing Address Mark) flag in Status Register 1 is set to a 1 (high), and if no data is found then the ND (No Data) flag is also set in Status Register 1 to a 1 (high) and the command is terminated.

FORMAT A TRACK

The Format Command allows an entire track to be formatted. After the INDEX HOLE is detected, Data is written on the Diskette: Gaps, Address Marks, ID Fields and Data Fields, all per the IBM System 34 (Double Density) or System 3740 (Single Density) Format are recorded. The particular format which will be written is controlled by the values programmed into N (number of bytes/sector), SC (sectors/cylinder), GPL (Gap Length), and D (Data Pattern) which are supplied by the processor during the Command Phase. The Data Field is filled with the Byte of data stored in D. The ID Field for each sector is supplied by the processor; that is, four data requests per sector are made by the FDC for C (Cylinder Number), H(Head Number), R(Sector Number) and N(Number of Bytes/Sector). This allows the diskette to be formatted with nonsequential sector numbers, if desired.

After formatting each sector, the processor must send new values for C, H, R, and N to the 8272A for each sector on the track. The contents of the R Register is incremented by one after each sector is formatted, thus, the R register contains a value of R + 1 when it is read during the Result Phase. This incrementing and formatting continues for the whole track until the FDC encounters the INDEX HOLE for the second time, whereupon it terminates the command.

If a FAULT signal is received from the FDD at the end of a write operation, then the FDC sets the EC flag of Status Register 0 to a 1 (high), and terminates the command after setting bits 7 and 6 of Status Register 0 to 0 and 1 respectively. Also the loss of a READY signal at the beginning of a command execution phase causes command termination.

Table 9 shows the relationship between N, SC, and GPL for various sector sizes:

Table 9. Sector Size Relationships

Format	Bytes/ Sector	8" Floppy				Bytes/ Sector	5¼" Floppy				Bytes/ Sector	3½" Mini Floppy			
		N	SC	GPL(1)	GPL(2)		N	SC	GPL(1)	GPL(2)		N	SC	GPL(1)	GPL(2)
FM Mode	128	00	1A	07	1B	128	00	12	07	09	128	0	0F	07	1B
	256	01	0F	0E	2A	128	00	10	10	19	—	—	—	—	—
	512	02	08	1B	3A	256	01	08	18	30	256	1	09	0F	2A
	1024	03	04	47	8A	512	02	04	46	87	512	2	05	1B	3A
	2048	04	02	C8	FF	1024	03	02	C8	FF	—	—	—	—	—
	4096	05	01	C8	FF	2048	04	01	C8	FF	—	—	—	—	—
MPM Mode	256	01	1A	0E	36	256	01	12	0A	0C	256	1	0F	CE	36
	512	02	OF	1B	54	256	01	10	20	32	—	—	—	—	—
	1024	03	08	35	74	512	02	08	2A	50	512	2	09	1B	54
	2048	04	04	99	FF	1024	03	04	80	F0	1024	3	05	35	74
	4096	05	02	C8	FF	2048	04	02	C8	FF	—	—	—	—	—
	8192	06	01	C8	FF	4096	05	01	C8	FF	—	—	—	—	—

NOTES:
1. Suggested values of GPL in Read or Write Commands to avoid splice point between data field and ID field of contiguous sections.
2. Suggested values of GPL in format command.

SCAN COMMANDS

The SCAN Commands allow data which is being read from the diskette to be compared against data which is being supplied from the main system (Processor in NON-DMA mode, and DMA Controller in DMA mode). The FDC compares the data on a byte-by-byte basis, and looks for a sector of data which meets the conditions of $D_{FDD} = D_{Processor}$, $D_{FDD} \leq D_{Processor}$, or $D_{FDD} \geq D_{Processor}$. Ones complement arithmetic is used for comparison (FF = largest number, 00 = smallest number). After a whole sector of data is compared, if the conditions are not met, the sector number is incremented (R + STP → R), and the scan operation is continued. The scan operation continues until one of the following conditions occur; the conditions for scan are met (equal, low, or high), the last sector on the track is reached (EOT), or the terminal count signal is received.

If the conditions for scan are met then the FDC sets the SH (Scan Hit) flag of Status Register 2 to a 1 (high), and terminates the Scan Command. If the

Table 10. Scan Status Codes

Command	Status Register 2		Comments
	Bit 2 = SN	Bit 3 = SH	
Scan Equal	0	1	$D_{FDD} = D_{Processor}$
	1	0	$D_{FDD} \neq D_{Processor}$
Scan Low or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} < D_{Processor}$
	1	0	$D_{FDD} \not> D_{Processor}$
Scan High or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} > D_{Processor}$
	1	0	$D_{FDD} \not< D_{Processor}$

conditions for scan are not met between the starting sector (as specified by R) and the last sector on the cylinder (EOT), then the FDC sets the SN (Scan Not Satisfied) flag of Status Register 2 to a 1 (high), and terminates the Scan Command. The receipt of a TERMINAL COUNT signal from the Processor or DMA Controller during the scan operation will cause the FDC to complete the comparison of the particular byte which is in process, and then to terminate the command. Table 10 shows the status of bits SH and SN under various conditions of SCAN.

If the FDC encounters a Deleted Data Address Mark on one of the sectors (and SK = 0), then it regards the sector as the last sector on the cylinder, sets CM (Control Mark) flag of Status Register 2 to a 1 (high) and terminates the command. If SK = 1, the FDC skips the sector with the Deleted Address Mark, and reads the next sector. In the second case (SK = 1), the FDC sets the CM (Control Mark) flag of Status Register 2 to a 1 (high) in order to show that a Deleted Sector had been encountered.

When either the STP (contiguous sectors STP = 01, or alternate sectors STP = 02 sectors are read) or the MT (Multi-Track) are programmed, it is necessary to remember that the last sector on the track must be read. For example, if STP = 02, MT = 0, the sectors are numbered sequentially 1 through 26, and we start the Scan Command at sector 21; the following will happen. Sectors 21, 23, and 25 will be read, then the next sector (26) will be skipped and the Index Hole will be encountered before the EOT value of 26 can be read. This will result in an abnormal termination of the command. If the EOT had been set at 25 or the scanning started at sector 20, then the Scan Command would be completed in a normal manner.

During the Scan Command data is supplied by either the processor or DMA Controller for comparison against the data read from the diskette. In order to avoid having the OR (Over Run) flag set in Status Register 1, it is necessary to have the data available in less than 27 μs (FM Mode) or 13 μs (MFM Mode). If an Overrun occurs the FDC terminates the command.

SEEK

The read/write within the FDD is moved from cylinder to cylinder under control of the Seek Command. The FDC compares the PCN (Present Cylinder Number) which is the current head position with the NCN (New Cylinder Number), and performs the following operation if there is a difference:

PCN < NCN: Direction signal to FDD set to a 1 (high), and Step Pulses are issued. (Step In.)

PCN > NCN: Direction signal to FDD set to a 0 (low), and Step Pulses are issued. (Step Out.)

The rate at which Step Pulses are issued is controlled by SRT (Stepping Rate Time) in the SPECIFY Command. After each Step Pulse is issued NCN is compared against PCN, and when NCN = PCN, then the SE (Seek End) flag is set in Status Register 0 to a 1 (high), and the command is terminated.

During the Command Phase of the Seek operation the FDC is in the FDC BUSY state, but during the Execution Phase it is in the NON BUSY state. While the FDC is in the NON BUSY state, another Seek Command may be issued, and in this manner parallel seek operations may be done on up to 4 Drives at once.

If an FDD is in a NOT READY state at the beginning of the command execution phase or during the seek operation, then the NR (NOT READY) flag is set in Status Register 0 to a 1 (high), and the command is terminated.

Note that the 8272A Read and Write Commands do not have implied Seeks. Any R/W command should be preceded by: 1) Seek Command; 2) Sense Interrupt Status; and 3) Read ID.

RECALIBRATE

This command causes the read/write head within the FDD to retract to the Track 0 position. The FDC clears the contents of the PCN counter, and checks the status of the Track 0 signal from the FDD. As long as the Track 0 signal is low, the Direction signal remains 1 (high) and Step Pulses are issued. When the Track 0 signal goes high, the SE (SEEK END) flag in Status Register 0 is set to a 1 (high) and the command is terminated. If the Track 0 signal is still low after 77 Step Pulses have been issued, the FDC sets the SE (SEEK END) and EC (EQUIPMENT CHECK) flags of Status Register 0 to both 1s (highs), and terminates the command.

The ability to overlap RECALIBRATE Commands to multiple FDDs, and the loss of the READY signal, as described in the SEEK Command, also applies to the RECALIBRATE Command.

SENSE INTERRUPT STATUS

An Interrupt signal is generated by the FDC for one of the following reasons:

1) Upon entering the Result Phase of:
   a) Read Data Command
   b) Read a Track Command
   c) Read ID Command
   d) Read Deleted Data Command
   e) Write Data Command
   f) Format a Cylinder Command
   g) Write Deleted Data Command
   h) Scan Commands
2) Ready Line of FDD changes state
3) End of Seek or Recalibrate Command
4) During Execution Phase in the NON-DMA Mode

Interrupts caused by reasons 1 and 4 above occur during normal command operations and are easily discernible by the processor. However, interrupts caused by reasons 2 and 3 above may be uniquely identified with the aid of the Sense Interrupt Status Command. This command when issued resets the interrupt signal and via bits 5, 6, and 7 of Status Register 0 identifies the cause of the interrupt.

Neither the Seek or Recalibrate Command have a Result Phase. Therefore, it is mandatory to use the Sense Interrupt Status Command after these commands to effectively terminate them and to provide verification of the head position (PCN).

Table 11. Seek, Interrupt Codes

Seek End Bit 5	Interrupt Code		Cause
	Bit 6	Bit 7	
0	1	1	Ready Line Changed State, Either Polarity
1	0	0	Normal Termination of Seek or Recalibrate Command
1	1	0	Abnormal Termination of Seek or Recalibrate Command

SPECIFY

The Specify Command sets the intitial values for each of the three internal timers. The HUT (Head Unload Time) defines the time from the end of the Execution Phase of one of the Read/Write Commands to the head unload state. This timer is programmable from 16 to 240 ms in increments of 16 ms (01 = 16 ms, 02 = 32 ms . . . . OF = 240 ms). The SRT (Step Rate Time) defines the time interval between adjacent step pulses. This timer is programmable from 1 to 16 ms in increments of 1 ms (F = 1 ms, E = 2 ms, D = 3 ms, etc.). The HLT (Head Load Time) defines the time between when the Head Load signal goes high and when the Read/Write operation starts. This timer is programmable from 2 to 254 ms in increments of 2 ms (01 = 2 ms, 02 = 4 ms, 03 = 6 ms . . . . FE = 254 ms).

The step rate should be programmed 1 ms longer than the minimum time required by the drive.

The time intervals mentioned above are a direct function of the clock (CLK on pin 19). Times indicated above are for an 8 MHz clock, if the clock was reduced to 4 MHz (mini-floppy application) then all time intervals are increased by a factor of 2.

The choice of DMA or NON-DMA operation is made by the ND (NON-DMA) bit. When this bit is high (ND = 1) the NON-DMA mode is selected, and when ND = 0 the DMA mode is selected.

SENSE DRIVE STATUS

This command may be used by the processor whenever it wishes to obtain the status of the FDDs. Status Register 3 contains the Drive Status information.

INVALID

If an invalid command is sent to the FDC (a command not defined above), then the FDC will terminate the command. No interrupt is generated by the 8272A during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both high ("1") indicating to the processor that the 8272A is in the Result Phase and the contents of Status Register 0 (ST0) must be read. When the processor reads Status Register 0 it will find an 80H indicating an invalid command was received.

A Sense Interrupt Status Command must be sent after a Seek or Recalibrate interrupt, otherwise the FDC will consider the next command to be an Invalid Command.

In some applications the user may wish to use this command as a No-Op command, to place the FDC in a standby or no operation state.

Table 12. Status Registers

Bit No.	Bit Name	Bit Symbol	Description
STATUS REGISTER 0			
D_7	Interrupt Code	IC	$D_7 = 0$ and $D_6 = 0$ Normal Termination of Command, (NT). Command was completed and properly executed.
D_6			$D_7 = 0$ and $D_6 = 1$ Abnormal Termination of Command, (AT). Execution of Command was started, but was not successfully completed.
			$D_7 = 1$ and $D_6 = 0$ Invalid Command issue, (IC). Command which was issued was never started.
			$D_7 = 1$ and $D_6 = 1$ Abnormal Termination because during command execution the ready signal from FDD changed state.
D_5	Seek End	SE	When the FDC completes the SEEK Command, this flag is set to 1 (high).
D_4	Equipment Check	EC	If a fault Signal is received from the FDD, or if the Track 0 Signal fails to occur after 77 Step Pulses (Recalibrate Command) then this flag is set.
D_3	Not Ready	NR	When the FDD is in the not-ready state and a read or write command is issued, this flag is set. If a read or write command is issued to Side 1 of a single sided drive, then this flag is set.
D_2	Head Address	HD	This flag is used to indicate the state of the head at Interrupt.
D_1	Unit Select 1	US 1	These flags are used to indicate a Drive Unit Number at Interrupt.
D_0	Unit Select 0	US 0	
STATUS REGISTER 1			
D_7	End of Cylinder	EN	When the FDC tries to access a Sector beyond the final Sector of a Cylinder, this flag is set.
D_6			Not used. This bit is always 0 (low).
D_5	Data Error	DE	When the FDC detects a CRC error in either the ID field or the data field, this flag is set.
D_4	Over Run	OR	If the FDC is not serviced by the main-systems during data transfers, within a certain time interval, this flag is set.
D_3			Not used. This bit always 0 (low).
D_2	No Data	ND	During execution of READ DATA, WRITE DELETED DATA or SCAN Command, if the FDC cannot find the Sector specified in the IDR Register, this flag is set.
			During executing the READ ID Command, if the FDC cannot read the ID field without an error, then this flag is set.
			During the execution of the READ A Cylinder Command, if the starting sector cannot be found, then this flag is set.

Table 12. Status Register (Continued)

Bit No.	Bit Name	Bit Symbol	Description
STATUS REGISTER 1 (Continued)			
D_1	Not Writable	NW	During execution of WRITE DATA, WRITE DELETED DATA or Format A Cylinder Command, if the FDC detects a write protect signal from the FDD, then this flag is set.
D_0	Missing Address Mark	MA	If the FDC cannot detect the ID Address Mark after encountering the index hole twice, then this flag is set.
			If the FDC cannot detect the Data Address Mark or Deleted Data Address Mark, this flag is set. Also at the same time, the MD (Missing Address Mark in Data Field) of Status Register 2 is set.
STATUS REGISTER 2			
D_7			Not used. This bit is always 0 (low).
D_6	Control Mark	CM	During executing the READ DATA or SCAN Command, if the FDC encounters a Sector which contains a Deleted Data Address Mark, this flag is set.
D_5	Data Error in Data Field	DD	If the FDC detects a CRC error in the data field then this flag is set.
D_4	Wrong Cylinder	WC	This bit is related with the ND bit, and when the contents of C on the medium is different from that stored in the IDR, this flag is set.
D_3	Scan Equal Hit	SH	During execution, the SCAN Command, if the condition of "equal" is satisfied, this flag is set.
D_2	Scan Not Satisfied	SN	During executing the SCAN Command, if the FDC cannot find a Sector on the cylinder which meets the condition, then this flag is set.
D_1	Bad Cylinder	BC	This bit is related with the ND bit, and when the content of C on the medium is different from that stored in the IDR and the content of C is FF, then this flag is set.
D_0	Missing Address Mark in Data Field	MD	When data is read from the medium, if the FDC cannot find a Data Address Mark or Deleted Data Address Mark, then this flag is set.
STATUS REGISTER 3			
D_7	Fault	FT	This bit is used to indicate the status of the Fault signal from the FDD.
D_6	Write Protected	WP	This bit is used to indicate the status of the Write Protected signal from the FDD.
D_5	Ready	RDY	This bit is used to indicate the status of the Ready signal from the FDD.
D_4	Track 0	T0	This bit is used to indicate the status of the Track 0 signal from the FDD.
D_3	Two Side	TS	This bit is used to indicate the status of the Two Side signal from the FDD.
D_2	Head Address	HD	This bit is used to indicate the status of Side Select signal to the FDD.
D_1	Unit Select 1	US 1	This bit is used to indicate the status of the Unit Select 1 signal to the FDD.
D_0	Unit Select 0	US 0	This bit is used to indicate the status of the Unit Select 0 signal to the FDD.

ABSOLUTE MAXIMUM RATINGS*

Operating Temperature .....0°C to +70°C
Storage Temperature .....−40°C to +125°C
All Output Voltages .....−0.5 to +7V
All Input Voltages .....−0.5 to +7V
Supply Voltage V_{CC} .....−0.5 to +7V
Power Dissipation .....1 Watt
*T_A = 25°C

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = +5V ±10%

Symbol	Parameter	Limits		Unit	Test Conditions
		Min	Max		
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4	V_{CC}	V	I_{OH} = −400 μA
I_{CC}	V_{CC} Supply Current		120	mA	
I_{IL}	Input Load Current (All Input Pins)		10 −10	μA μA	V_{IN} = V_{CC} V_{IN} = 0V
I_{LOH}	High Level Output Leakage Current		10	μA	V_{OUT} = V_{CC}
I_{OFL}	Output Float Leakage Current		±10	μA	0.45C ≤ V_{OUT} ≤ V_{CC}

CAPACITANCE T_A = 25°C, f_c = 1 MHz, V_{CC} = 0V

Symbol	Parameter	Limits		Unit	Test Conditions
		Min	Max		
$C_{IN(\phi)}$	Clock Input Capacitance		20	pF	All Pins Except Pin Under Test Tied to AC Ground
C_{IN}	Input Capacitance		10	pF	
$C_{I/O}$	Input/Output Capacitance		20	pF	

A.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$, $V_{CC} = +5.0V \pm 10\%$

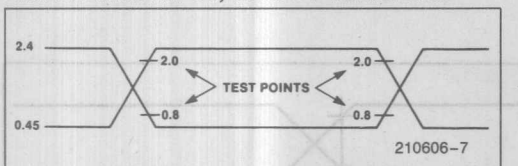
Symbol	Parameter	Typ(1)	Min	Max	Unit	Notes
CLOCK TIMING						
t_{CY}	Clock Period		120	500	ns	(Note 5)
t_{CH}	Clock High Period		40		ns	(Note 4, 5)
t_{RST}	Reset Width		14		t_{CY}	
READ CYCLE						
t_{AR}	Select Setup to $\overline{RD}\downarrow$		0		ns	
t_{RA}	Select Hold from $\overline{RD}\uparrow$		0		ns	
t_{RR}	$\overline{RD}$ Pulse Width		250		ns	
t_{RD}	Data Delay from $\overline{RD}\downarrow$			200	ns	
t_{DF}	Output Float Delay		20	100	ns	
WRITE CYCLE						
t_{AW}	Select Setup to $\overline{WR}\downarrow$		0		ns	
t_{WA}	Select Hold from $\overline{WR}\uparrow$		0		ns	
t_{WW}	$\overline{WR}$ Pulse Width		250		ns	
t_{DW}	Data Setup to $\overline{WR}\uparrow$		150		ns	
t_{WD}	Data Hold from $\overline{WR}\uparrow$		5		ns	
INTERRUPTS						
t_{RI}	INT Delay from $\overline{RD}\uparrow$			500	ns	(Note 6)
t_{WI}	INT Delay from $\overline{WR}\uparrow$			500	ns	(Note 6)
DMA						
t_{RQCY}	DRQ Cycle Period		13		μs	(Note 6)
t_{AKRQ}	$\overline{DACK}\downarrow$ to DRQ $\downarrow$			200	ns	
t_{RQR}	DRQ $\uparrow$ to $\overline{RD}\downarrow$		800		ns	(Note 6)
t_{RQW}	DRQ $\uparrow$ to $\overline{WR}\downarrow$		250		ns	(Note 6)
t_{RQRW}	DRQ $\uparrow$ to $\overline{RD}\uparrow$ or $\overline{WR}\uparrow$			12	μs	(Note 6)

A.C. CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = +5.0V ±10% (Continued)

Symbol	Parameter	Typ(1)	Min	Max	Unit	Notes
FDD INTERFACE						
t_{WCY}	WCK Cycle Time	2 or 4 1 or 2			µs	MFM = 0 MFM = 1 (Note 2)
t_{WCH}	WCK High Time	250	80	350	ns	
t_{CP}	Pre-Shift Delay from WCK ↑		20	100	ns	
t_{CD}	WDA Delay from WCK ↑		20	100	ns	
t_{WDD}	Write Data Width		t_{WCH} − 50		ns	
t_{WE}	WE ↑ to WCK ↑ or WE ↓ to WCK ↓ Delay		20	100	ns	
t_{WWCY}	Window Cycle Time	2 1			µs	MM = 0 MFM = 1
t_{WRD}	Window Setup to RDD ↑		15		ns	
t_{RDW}	Window Hold from RDD ↓		15		ns	
t_{RDD}	RDD Active Time (HIGH)		40		ns	
FDD SEEK/DIRECTION/STEP						
t_{US}	$US_{0,1}$ Setup to $\overline{RW}$/SEEK ↑		12		µs	(Note 6)
t_{SU}	$US_{0,1}$ Hold after $\overline{RW}$/SEEK ↓		15		µs	(Note 6)
t_{SD}	$\overline{RW}$/SEEK Setup to LCT/DIR		7		µs	(Note 6)
t_{DS}	$\overline{RW}$/SEEK Hold from LCT/DIR		30		µs	(Note 6)
t_{DST}	LCT/DIR Setup to FR/STEP ↑		1		µs	(Note 6)
t_{STD}	LCT/DIR Hold from FR/STEP ↓		24		µs	(Note 6)
t_{STU}	$DS_{2,1}$ Hold from FR/Step ↓		5		µs	(Note 6)
t_{STP}	STEP Active Time (High)	5			µs	(Note 6)
t_{SC}	STEP Cycle Time		33		µs	(Note 3, 6)
t_{FR}	FAULT RESET Active Time (High)		8	10	µs	(Note 6)
t_{IDX}	INDEX Pulse Width	10			t_{CY}	
t_{TC}	Terminal Count Width		1		t_{CY}	

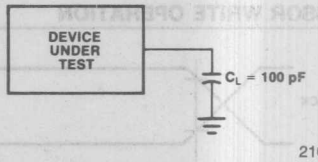
NOTES:
1. Typical values for T_A = 25°C and nominal supply voltage.
2. The former values are used for standard floppy and the latter values are used for mini-floppies.
3. t_{SC} = 33 µs min. is for different drive units. In the case of same unit, t_{SC} can be ranged from 1 ms to 16 ms with 8 MHz clock period, and 2 ms to 32 ms with 4 MHz clock, under software control.
4. From 2.0V to +2.0V.
5. At 4 MHz, the clock duty cycle may range from 16% to 76%. Using an 8 MHz clock the duty cycle can range from 32% to 52%. Duty cycle is defined as: D.C. = 100 (t_{CH} ÷ t_{CY}) with typical rise and fall times of 5 ns.
6. The specified values listed are for an 8 MHz clock period. Multiply timings by 2 when using a 4 MHz clock period.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

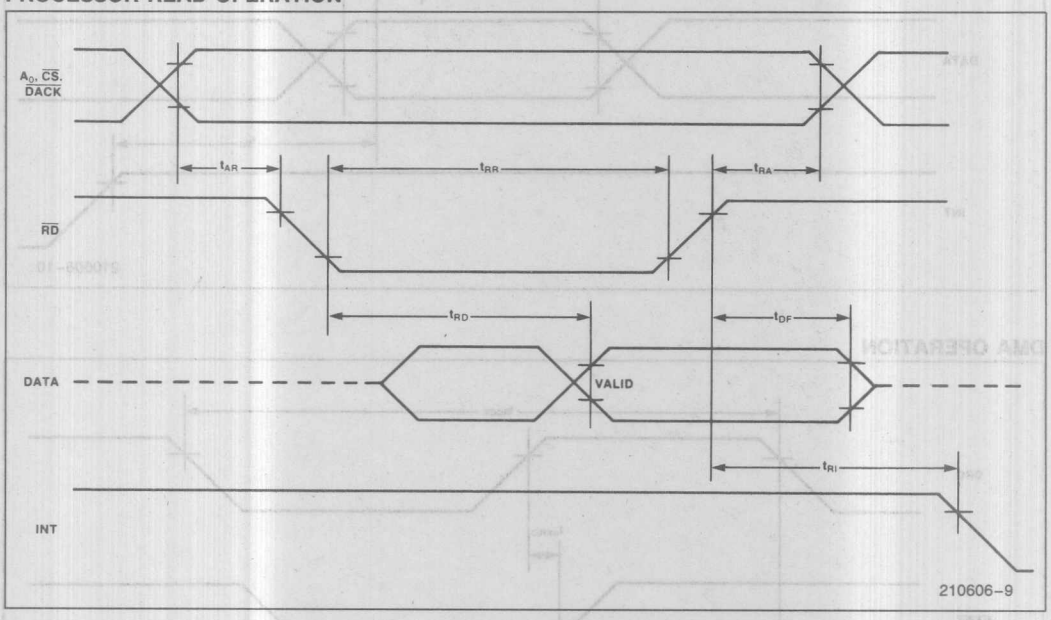
A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L Includes Jig Capacitance

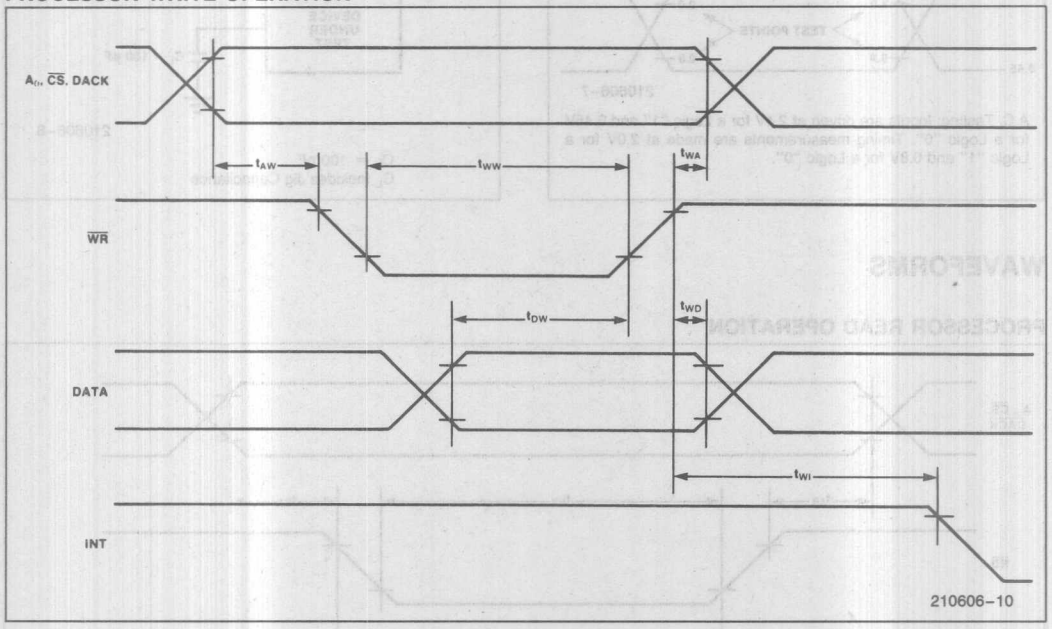
WAVEFORMS

PROCESSOR READ OPERATION

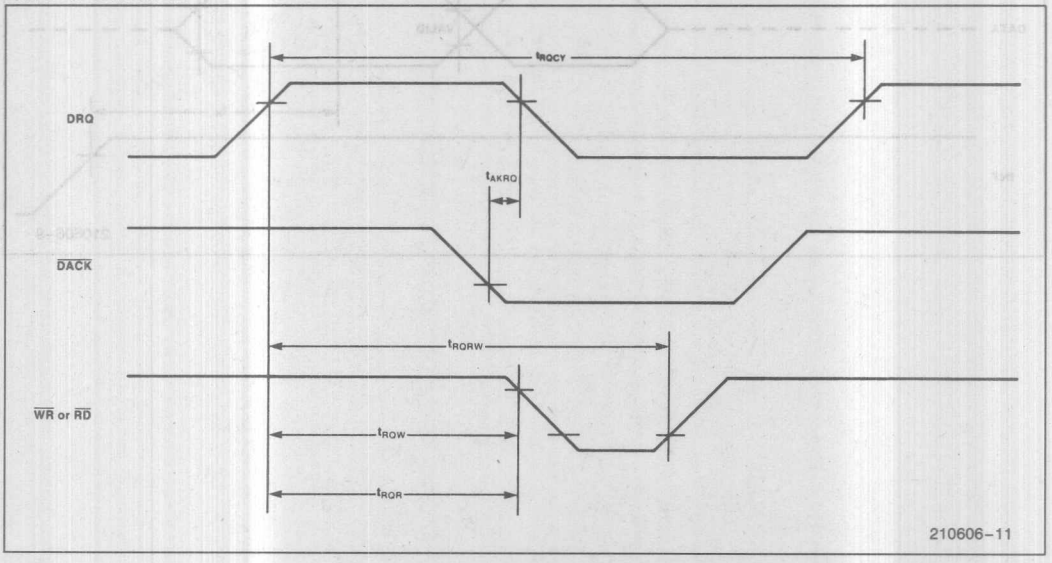


WAVEFORMS (Continued)

PROCESSOR WRITE OPERATION

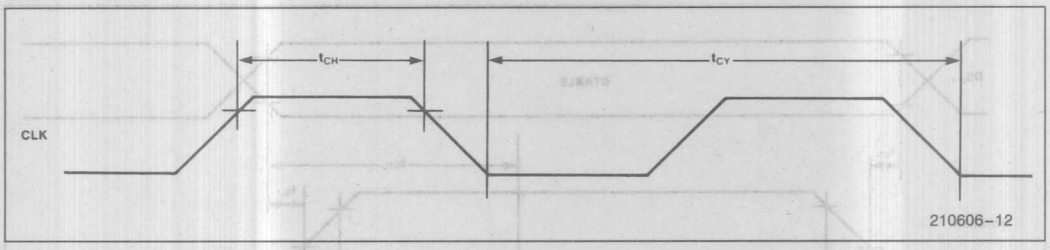


DMA OPERATION

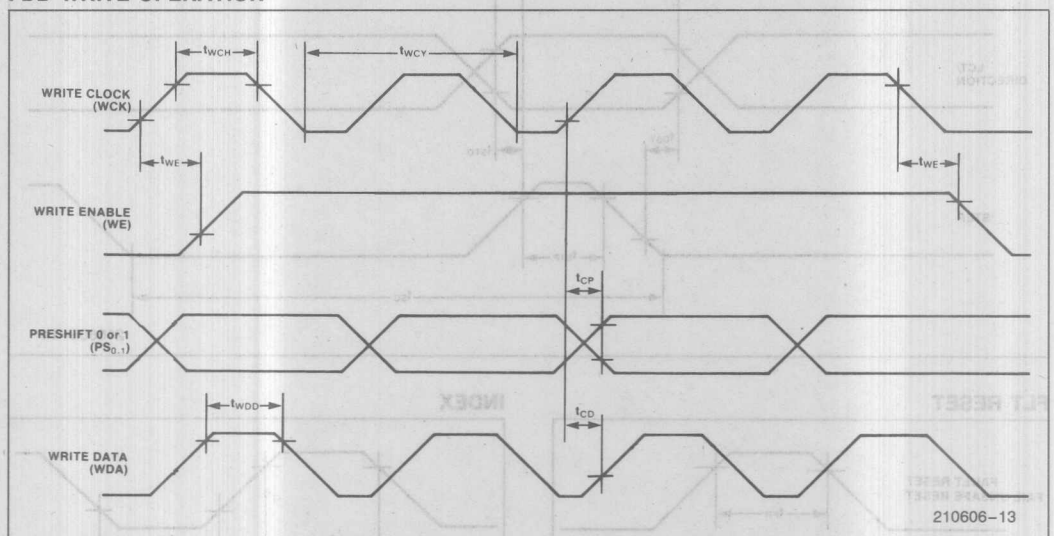


WAVEFORMS (Continued)

CLOCK TIMING



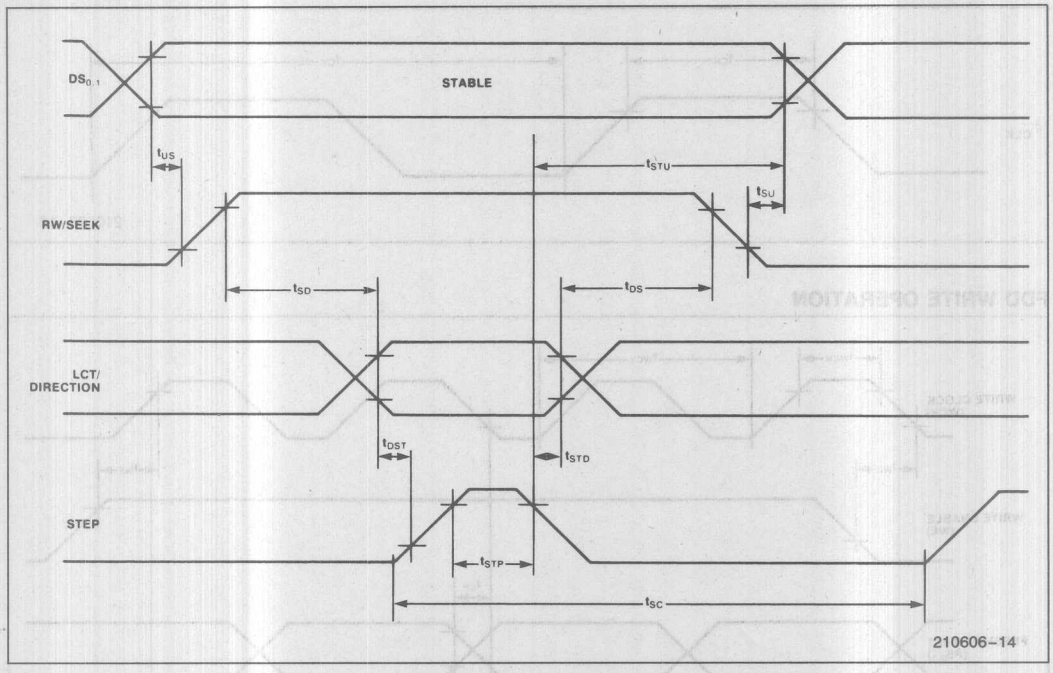
FDD WRITE OPERATION



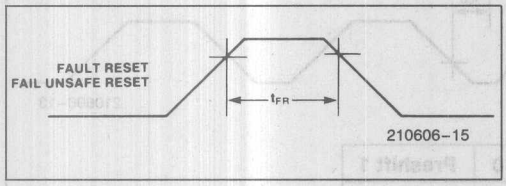
	Preshift 0	Preshift 1
Normal	0	0
Late	0	1
Early	1	0
Invalid	1	1

WAVEFORMS (Continued)

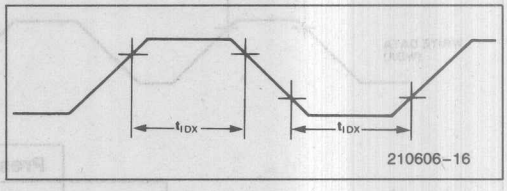
SEEK OPERATION



FLT RESET

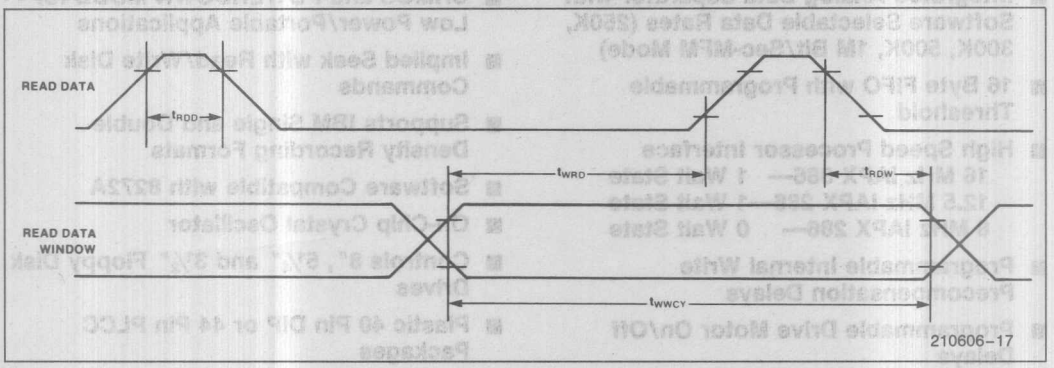


INDEX

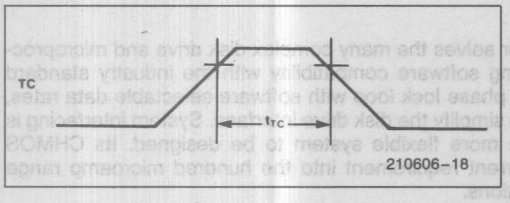


WAVEFORMS (Continued)

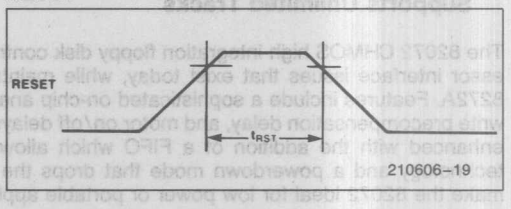
FDD READ OPERATION



TERMINAL COUNT



RESET



82072
CHMOS HIGH INTEGRATION FLOPPY DISK CONTROLLER

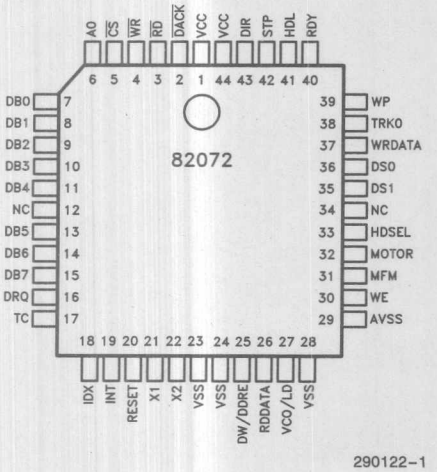
- **Integrated Analog Data Separator with Software Selectable Data Rates (250K, 300K, 500K, 1M Bit/Sec-MFM Mode)**
- **16 Byte FIFO with Programmable Threshold**
- **High Speed Processor Interface**
  - **16 MHz iAPX 386— 1 Wait State**
  - **12.5 MHz iAPX 286—1 Wait State**
  - **8 MHz iAPX 286— 0 Wait State**
- **Programmable Internal Write Precompensation Delays**
- **Programmable Drive Motor On/Off Delays**
- **Addresses Up to 256 Tracks Directly, Supports Unlimited Tracks**
- **CHMOS and POWERDOWN MODE for Low Power/Portable Applications**
- **Implied Seek with Read/Write Disk Commands**
- **Supports IBM Single and Double Density Recording Formats**
- **Software Compatible with 8272A**
- **On-Chip Crystal Oscillator**
- **Controls 8", 5¼" and 3½" Floppy Disk Drives**
- **Plastic 40 Pin DIP or 44 Pin PLCC Packages**

(See Packaging Spec. Order #231369)

The 82072 CHMOS high integration floppy disk controller solves the many complex disk drive and microprocessor interface issues that exist today, while maintaining software compatibility with the industry standard 8272A. Features include a sophisticated on-chip analog phase lock loop with software selectable data rates, write precompensation delay, and motor on/off delays to simplify the disk drive interface. System interfacing is enhanced with the addition of a FIFO which allows a more flexible system to be designed. Its CHMOS technology and a powerdown mode that drops the current requirement into the hundred microamp range make the 82072 ideal for low power or portable applications.

The standard 82072 supports a maximum data rate of 500 Kbits per second. The 82072-1 supports data rates up to 1 Mbit per second.

The 82072 is fabricated on Intel's advanced CHMOS III technology for minimal power consumption and is available in a plastic 40 pin DIP or a plastic 44-leaded chip carrier (PLCC) package.



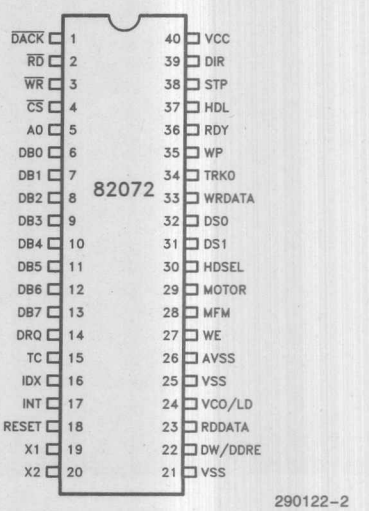


Figure 1. 82072 Pinout

October 1987
Order Number: 290122-004

Table 1. 82072 Pin Description

Symbol	DIP	PLCC	I/O	Function
$\overline{DACK}$	1	2	I	**DMA ACKNOWLEDGE:** DMA control line that qualifies the $\overline{RD}$ and $\overline{WR}$ inputs during DMA cycles.
$\overline{RD}$	2	3	I	**READ:** Control signal to transfer data to the data bus from the 82072.
$\overline{WR}$	3	4	I	**WRITE:** Control signal to transfer data into the 82072 from the data bus.
$\overline{CS}$	4	5	I	**CHIP SELECT:** Control signal that qualifies the $\overline{RD}$ and $\overline{WR}$ inputs.
A0	5	6	I	**ADDRESS:** A0 $\overline{RD}$ $\overline{WR}$ Function 0 0 0 Illegal * 0 0 1 Read Main Status Register 0 1 0 Write to the Data Rate Select Register ** 0 1 1 No Action 1 0 0 Illegal * 1 0 1 Read from FIFO 1 1 0 Write into FIFO 1 1 1 No Action * User must ensure that these inputs do not occur. ** Change from 8272A—was illegal.
DB0–7	6–13	7–11, 13–15	I/O	**DATA BUS:** Bidirectional 8-bit data bus. A0 determines whether transfer is to/from the FIFO or Main Status Register.
DRQ	14	16	O	**DMA REQUEST:** Used to request service from a DMA controller.
TC	15	17	I	**TERMINAL COUNT:** Control line from a DMA controller used to terminate requests for data transfers. Disk read and write commands complete the transfer to the current sector with valid CRC checking/generation.
IDX	16	18	I	**INDEX:** Disk drive signal that indicates the beginning of a track. It is used to count retries and delay periods for internal (i.e. Motor On/Off) timers and is rising edge triggered.
INT	17	19	O	**Interrupt:** Interrupt to host to indicate command completion or that a data transfer is required (depending upon the data transfer mode). Command completion interrupts are cleared by reading the ST0 Status Register. Data transfer interrupts are cleared when the amount of data in the FIFO reaches the full or empty level (depending on FIFO direction) or a TC is issued.
RESET	18	20	I	**RESET:** Places the 82072 in a known idle state. All disk outputs are set to a low level. All registers, except those set by the SPECIFY command, are cleared. From the trailing edge of Reset, there is a maximum delay of 8 microseconds until the Main Status register is valid. Following reset, the 82072 defaults to polling enabled. The default values for the new features are: internal data separator enabled, write precompensation value is 125 ns, MOTOR on delay is 0.0 sec., MOTOR off delay is 5.2 sec., data rate is dependent on DDRE setting, FIFO disabled.

Table 1. 82072 Pin Description (Continued)

Symbol	DIP	PLCC	I/O	Function
X1	19	21	I	**CRYSTAL 1:** External connection for a fundamental mode parallel resonant 24 MHz crystal for the internal oscillator. May be driven with a MOS level clock instead of a crystal. Refer to the D.C. Specifications.
X2	20	22	I	**CRYSTAL 2:** If an external clock is supplied on X1, this input must be left unconnected (floating).
V_{SS}	21, 25	23, 24, 28		**LOGIC GROUND.**
DW/DDRE	22	25	I	**DATA WINDOW/DEFAULT DATA RATE ENABLED:** Clock from the external PLL logic used to sample the Read Data input. When the internal PLL is used, this input pin is used to define the data rate and write precompensation values after RESET. DDRE tied high will cause the data rate and precompensation bits of the DSR to be reinitialized to the default values of 250 Kbps and 125 ns delay when a hardware/software reset is issued. DDRE tied low will cause the current data rate and precompensation values in the DSR to be retained when a hardware reset is issued. When a software reset is issued, the DSR will contain those values written into the register. DDRE tied low should be used in applications where data rate and precompensation information needs to be retained regardless of chip reset.
RDDATA	23	26	I	**READ DATA:** Serial FM or MFM encoded data from the disk drive.
VCO/LD	24	27	O	**READ DATA GATE:** This active high output enables an external PLL to synchronize to Read Data input from the disk drive. **LOW DENSITY:** This active high output is used by quad density disk drives to modify Read/Write head and data channel characteristics. This signal is activated when internal PLL is enabled and a data transfer rate of 250 or 300 Kbps is chosen.
AVSS	26	29		Analog ground for the Data Separator. It is recommended that care be taken to keep AVSS as noise free as possible. A separate connection to the ground plane is suggested.
WE	27	30	O	**WRITE ENABLE:** Disk drive control signal that enables the head to write onto the disk.
MFM	28	31	O	**MFM MODE:** When an external PLL is used, this output selects between single and double density (FM and MFM) modes. 1 = MFM, 0 = FM mode.
MOTOR	29	32	O	**MOTOR ENABLE:** Output used to activate the drive motor on the selected drive. Delays are programmable. With one output, this pin must be qualified with the drive select logic to provide motor enables for each drive.
HDSEL	30	33	O	**HEAD SELECT:** Signal used to select one of two sides on the disk. A 0 = side 0, a 1 = side 1.
DS1, 0	31, 32	35, 36	O	**DRIVE SELECT:** These outputs select one of four disk drives. DS0, DS1 = 0, 0 will select drive 0.
WRDATA	33	37	O	**WRITE DATA:** FM or MFM encoded serial data to the disk drive. No external precompensation is required.
TRK0	34	38	I	**TRACK 0:** Control line from the disk drive that indicates the head is on physical track 0 (outermost track).
WP	35	39	I	**WRITE PROTECT:** Input from the disk drive that indicates if the disk is physically write protected.

Table 1. 82072 Pin Description (Continued)

Symbol	DIP	PLCC	I/O	Function
RDY	36	40	I	**READY:** Input from the disk drive that indicates whether the drive is ready for an operation.
HDL	37	41	O	**HEAD LOAD:** This output loads the head onto the disk drive if required. Typically used by 8" drives.
STP	38	42	O	**STEP:** Output used to supply step pulses to the disk drive.
DIR	39	43	O	**DIRECTION:** This output, in conjunction with STP, causes the drive to move the head outward if a "0", and inward if a "1".
VCC	40	1, 44		Logic DC power supply.

NOTE:
1. Pins 12, 34 of the 44 pin PLCC package are not connected.

INTRODUCTION

The 82072 has integrated all of the complex circuitry required to interface microprocessor systems with disk drives that comply with the IBM System 34 Double Density (MFM) format or the IBM 3740 single density format (FM). The 82072 is a superset of the 8272A. Control over the new features was accomplished by adding extra registers and commands to the 82072. The 82072 will function like the 8272A Floppy Disk Controller (FDC) after being reset, with the added features being set to 8272A compatible default values. When accessing the disk drives, the 82072 is programmed the same as the 8272A.

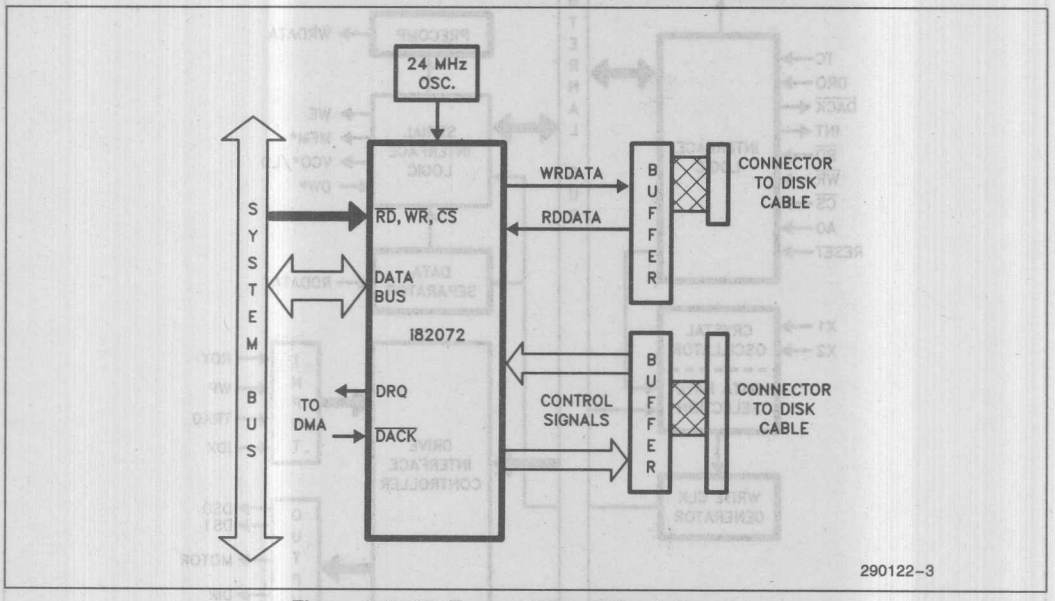


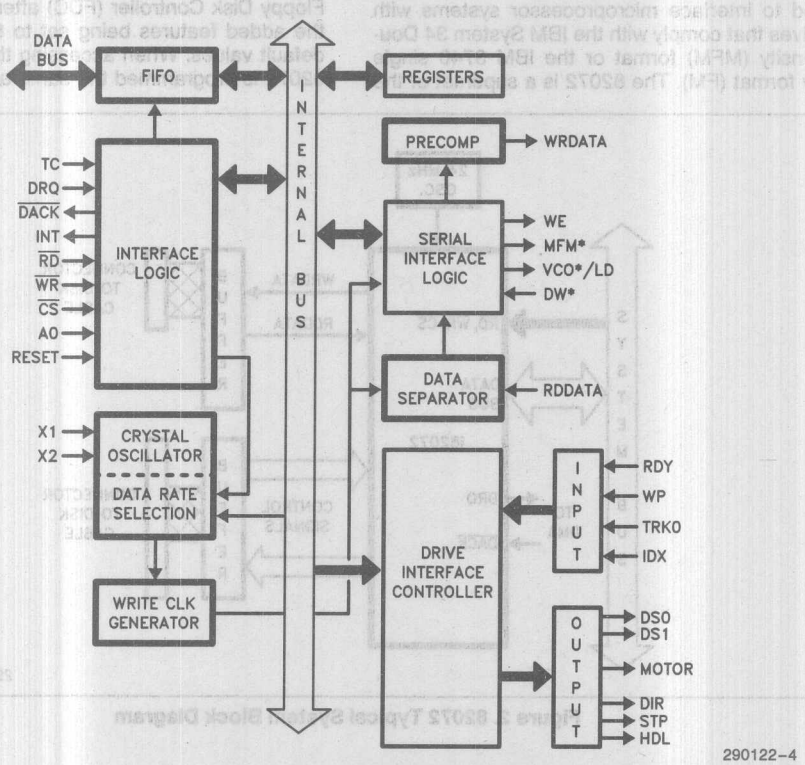
Figure 2. 82072 Typical System Block Diagram

The microprocessor interface was enhanced by adding a 16 byte FIFO to reduce the timing constraints that most floppy disk controllers impose upon a system. The point at which the 82072 generates a request for a data transfer is selectable within the 16 byte range of the FIFO. The interface was further enhanced to support todays faster microprocessors (i.e., 8 MHz 80286, 10 MHz 80186) without incurring wait states. A powerdown mode has been added for low power or portable applications. With one command, the 82072 resets itself and then disconnects the power from the internal oscillator. Reset will reconnect the clock and once the 82072 is reprogrammed (if necessary), it will be ready to read and write disks again.

All of the control logic of the disk interface has been integrated into the 82072. Flexibility is maintained by allowing the user to select read and write data rates (without any external hardware); write precompensation delays; motor on/off delay; and the track to start the precompensation on. The typical design will need the 82072, a crystal and high current drivers for the signals that interface to the disk drive. The new features (when used) need only to be chosen once after reset (although they may be modified at any time). From then on, the user programs the 82072 for disk accesses in the same manner as the 8272A.

ARCHITECTURE

Figure 3 is a block diagram of the 82072. The highlighted blocks represent areas that are either completely new or highly enhanced. All new features were adopted with the requirement of being software compatible with the 8272A.



*NOTE:
These pins are not used with the internal PLL

Figure 3. 82072 Block Diagram

Summary of Differences Between the 8272A and the 82072

Table 2 lists the features that are different between the 8272A and the 82072. The Scan commands are not supported by the 82072.

Table 2. Summary of Differences Between 82072 and 8272A

Features	82072	8272A
Process	CHMOS III	HMOS II
Data Separator	Internal	User Supplied
Data Rates	Up to 1 Mbit/sec. Software Selectable Without Hardware	Up to 500 kbit/sec External Hardware Needed
Power Down Mode	Internal	None
Commands	Superset of 8272A	
Host Interface	16 byte FIFO	1 byte Register
Software Reset	Yes	None
Implied Seeks	Yes	None
Relative Seek	Yes	None
Motor Delays	Yes	External Hardware
Write Precompensation Delays	Yes	External Hardware
Recalibrate	Issues 255 Step Pulses	Issues 77 Step Pulses

Crystal Oscillator Specification

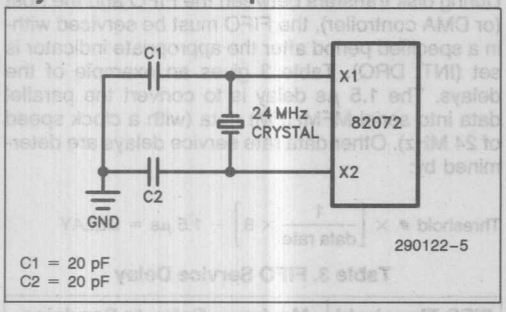


Figure 4. Crystal Oscillator Circuit

Specification of the parallel resonant crystal (typical values):

Frequency:	24 MHz
Mode:	Parallel resonant Fundamental mode
Series Resistance:	$R_S = 8\Omega$
Motional Inductance:	L = 2.2 mH
Motional Capacitance:	C = 0.02 pF
Shunt Capacitance:	C0 = 4.5 pF

The 24 MHz clock can be supplied either by a crystal or a MOS level square wave. All internal timings are referenced to this clock or a scaled count (which is determined by the data rate selection).

The crystal oscillator must be allowed to run for 10 mS after the VCC supply has reached 4.5 volts or 1 ms after the chip has been taken out of the Power-down mode before it is guaranteed to have stabilized.

MICROPROCESSOR INTERFACE

There are three ports accessible from the host's point of view; the FIFO, the Main Status Register (MSR) and the Data rate Select Register (DSR). Communication between the microprocessor and the 82072 is done by reading the Main Status register to determine if the controller is ready. If it is ready, a command, followed by the correct parameters, is sent to the 82072 through the FIFO (data port). The MSR can be read at any time and the DSR can be written at any time (before, during and after a command). The FIFO should only be accessed when the RQM bit in the MSR is set, or when DRQ is active if a DMA transfer is in progress.

MAIN STATUS REGISTER

RQM	DIO	NDM	CB	D3B	D2B	D1B	D0B
7	6	5	4	3	2	1	0

Bit 7—RQM (Request For Master)

This bit indicates that the host can access the FIFO (data port) if a "1". No accesses should be attempted if set to a "0".

Bit 6—DIO (Data In/Out)

Indicates the direction of the data transfer only when RQM is a "1". If DIO is a "1" the host should remove the bytes in the FIFO. A "0" means that the host should write into the FIFO.

Bit 5—NDM (Non-DMA Mode)

If the non-DMA mode is selected in the SPECIFY command, this bit will be set = "1" during the execution phase of a command. This bit is intended to support polled data transfers along with RQM and DIO. If DMA mode is selected, it remains a "0". This bit differentiates between the execution and result phases of a non-DMA READ DATA command.

Bit 4—CB (Controller Busy)

CB is set to a "1" when a command is in progress and indicates that the controller is processing a command. This bit will go active in the command phase after the command byte has been accepted. CB goes inactive at the end of the result phase which indicates the start of the next command phase. If the command has no result phase (i.e. SEEK and RECALIBRATE) the CB bit is cleared after receiving the last byte in the command phase.

Bit 0–3; Drive Busy

These bits are set only when a drive is in the seek portion of a command, including implied seeks, overlapped seeks and recalibrates.

FIFO

A 16 byte FIFO was added to increase the flexibility of a system when transferring data between the disk and memory by increasing the amount of time before service is required. This permits the 82072 to have a lower priority in servicing without losing data.

The 82072 FIFO is 16 bytes deep with the threshold set at any point (the threshold is set with the Configure command). The CONFIGURE command can enable the FIFO and set the FIFO threshold level. During writes into the FIFO (from the host), the 82072 will request service (via DRQ or INT-depending upon the selected transfer mode) when there are the programmed threshold number of bytes left in the FIFO. The request will go inactive once the FIFO is filled. When reading from the FIFO (to the host), service is requested when there are more than (16-threshold number) bytes in the FIFO. The request for service goes inactive when the FIFO is emptied. When the FIFO is disabled, a single byte per transfer protocol is enforced. The FIFO is still present and storage still occurs in each of the 16 bytes in sequence. Only the transfer protocol is changed.

During the Command phase (see next section), the FIFO action is disabled and data must be sent only after the RQM bit in the Main Status register has indicated that it is ready. Improper operation will result if command bytes are sent before the 82072 if ready. The 82072 does not ask for command parameters by generating interrupts or DMA requests. All command parameters are sent by polling the MSR.

As the 82072 enters the execution phase, it clears the FIFO of any data to ensure that invalid data is not written or read from the disk. During writes to the disk, if the host fails to respond in time, "00" will be written to the disk. During reads from the disk, the 82072 will not go into the Result phase until all data has been removed from the FIFO or a Terminal Count (TC) has been issued.

An overrun or underrun condition will terminate the transfer of data between the FIFO and the host. Writes to the disk will complete the current sector (using "00"s) and generate CRC. Reads will require the host to read all data bytes out of the FIFO so that the Result phase may begin. The proper error bit will be set in the Status register. A Terminal Count (TC) will always get the 82072 into the Result phase at the end of the current sector.

During the Result phase, the FIFO is disabled and the result bytes are read out one at a time.

During disk transfers between the FIFO and the host (or DMA controller), the FIFO must be serviced within a specified period after the appropriate indicator is set (INT, DRQ). Table 3 gives an example of the delays. The 1.5 μs delay is to convert the parallel data into serial MFM/(FM) data (with a clock speed of 24 MHz). Other data rate service delays are determined by:

$$\text{Threshold \#} \times \left[\frac{1}{\text{data rate}} \times 8\right] - 1.5\ \mu s = \text{DELAY}$$

Table 3. FIFO Service Delay

FIFO Threshold Examples	Maximum Delay to Servicing at 1 Mbps Data Rate
1 byte	$1 \times 8\ \mu s - 1.5\ \mu s = 6.5\ \mu s$
2 bytes	$2 \times 8\ \mu s - 1.5\ \mu s = 14.5\ \mu s$
8 bytes	$8 \times 8\ \mu s - 1.5\ \mu s = 70.5\ \mu s$
15 bytes	$15 \times 8\ \mu s - 1.5\ \mu s = 118.5\ \mu s$

FIFO Threshold Examples	Maximum Delay to Servicing at 500 Kbps Data Rate
1 byte	$1 \times 16\ \mu s - 1.5\ \mu s = 14.5\ \mu s$
2 bytes	$2 \times 16\ \mu s - 1.5\ \mu s = 30.5\ \mu s$
8 bytes	$8 \times 16\ \mu s - 1.5\ \mu s = 126.5\ \mu s$
15 bytes	$15 \times 16\ \mu s - 1.5\ \mu s = 238.5\ \mu s$

The FIFO defaults to FIFO disabled. If the FIFO is enabled, DRQ stays active until the FIFO is emptied/filled. If the FIFO is disabled with the EFIFO bit in the CONFIGURE command, individual DRQ's are issued. Refer to the section on DMA transfers for more detail.

COMMAND EXECUTION PHASES

For simplicity, command handling in the 82072 can be divided into three phases: command, execution and result. Each phase is described in the following paragraphs.

Command Phase

After a RESET, the 82072 enters the Command phase and is ready to accept a command from host. For each of the commands, a defined set of command code bytes and parameter bytes has to be written to the 82072 before the command phase is complete. (Please refer to the Command Description section). These bytes of data must occur in the order prescribed. For example:

SEEK Command

D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	1	1	1	1	1st byte to the 82072
0	0	0	0	0	HDS	DS1	DS0	2nd byte to the 82072
<	—	—	—	—	NCN	—	—	> 3rd byte to the 82072

Before writing to the 82072, the host must examine the RQM and DIO bits of the Main Status Register. RQM, DIO must be equal to "1" and "0", respectively before command bytes may be written. RQM is set false by the 82072 after each write cycle until the received byte is processed. The 82072 asserts RQM again to request each parameter byte of the command, unless an illegal command condition is detected. After the last parameter byte is received, RQM remains "0", and the 82072 automatically enters the next phase as defined by the command definition.

The FIFO is disabled during the command phase to retain compatibility with the 8272A, and to provide for the proper handling of the "invalid command" condition.

Execution Phase

All data transfers to or from the 82072 occur during the execution phase, which can proceed in DMA or non-DMA mode as indicated in the SPECIFY command.

After RESET, the FIFO is disabled. Each data byte is transferred by an INT or DRQ depending on the DMA mode. The CONFIGURE command can enable the FIFO and set the FIFO threshold value.

The following paragraphs detail the operation of the FIFO flow control. In these descriptions, <threshold> is defined as the number of bytes available to the 82072 when service is requested from the host, and ranges from 1 to 16. The parameter FIFOTHR which the user programs is one less, and ranges from 0 to 15.

A low threshold value (i.e. 2) results in longer periods of time between service requests, but requires faster servicing of the request, for both read and write cases. The host reads (writes) from (to) the FIFO until empty (full), then the transfer request goes inactive. The host must be very responsive to the service request. This is the desired case for use with a "fast" system.

A high value of threshold (i.e. 12) is used with a "sluggish" system by affording a long latency period after a service request, but results in more frequent service requests.

NON-DMA MODE, Transfers from the FIFO to the Host:

The INT pin and RQM bits in the Main Status Register are activated when the FIFO contains (16 – <threshold>) bytes, or the last bytes of a full sector transfer have been placed in the FIFO. The INT pin can be used for interrupt driven systems and RQM can be used for polled systems. The host must respond to the request by reading data from the FIFO. This process is repeated until the last byte is transferred out of the FIFO. The 82072 will deactivate the INT pin and RQM bit when the FIFO becomes empty.

NON-DMA MODE, Transfers from the Host to the FIFO:

The INT pin and RQM bit in the Main Status Register are activated upon entering the execution phase of data transfer commands. The host must respond to the request by writing data into the FIFO. The INT pin and RQM bit remain true until the FIFO becomes full. They are set true again when the FIFO has <threshold> bytes remaining in the FIFO. The 82072 will also deactivate the INT pin and RQM bit when TC goes active, indicating that no more data is required. The 82072 enters the results phase after the last byte is taken by the 82072 from the FIFO (i.e., FIFO empty condition).

DMA MODE, Transfers from the FIFO to the Host:

The 82072 activates the DRQ pin when the FIFO contains (16 – <threshold>) bytes, or the last byte of a full sector transfer has been placed in the FIFO. The DMA controller must respond to the request by reading data from the FIFO. The 82072 will deactivate the DRQ pin when the FIFO becomes empty. DRQ goes inactive after $\overline{DACK}$ goes active for the last byte of a data transfer (or on the active edge of $\overline{RD}$, on the last byte, if no edge is present on $\overline{DACK}$). A data underrun may occur if DRQ is not removed in time to prevent an unwanted cycle. Refer to the section on DMA Transfers for more detail.

DMA MODE, Transfers from the Host to the FIFO:

The 82072 activates the DRQ pin when entering the execution phase of the data transfer commands. The DMA controller must respond by activating the $\overline{DACK}$ and $\overline{WR}$ pins, and placing data in the FIFO. DRQ remains active until the FIFO becomes full. DRQ is again set true when the FIFO has <threshold> bytes remaining in the FIFO. The 82072 will also deactivate the DRQ pin when TC becomes true, indicating that no more data is required. DRQ goes inactive after $\overline{DACK}$ goes active for the last byte of a data transfer (or on the active edge of $\overline{WR}$ on the last byte, if no edge is present on $\overline{DACK}$). A data overrun may occur if DRQ is not removed in time to prevent an unwanted cycle. Refer to the section on DMA Transfers for more detail.

The 82072 supports terminal count explicitly through the TC pin and implicitly through the underrun/overrun and end-of-track (EOT) functions. For full sector transfers, the EOT parameter can define the last sector to be transferred in a single or multisector transfer. If the last sector to be transferred is a partial sector, the host can stop transferring the data in mid-sector, and the 82072 will continue to complete the sector as if a hardware TC was received. The only difference between these implicit functions and TC is that they return "abnormal termination" result status. Such status indications can be ignored if they were expected.

Note that when the host is sending data to the 82072's FIFO, the internal sector count will be complete when the 82072 reads the last byte from its side of the FIFO. There may be a delay in the removal of the transfer request signal of up to the time taken for the 82072 to read the last 16 bytes from the FIFO. The host must tolerate this delay.

If before the execution of a command, the RDY pin is "0", the 82072 will set the corresponding bit in Status Register 0, and terminate the command phase. When an external TC signal is received, the external transfer is suspended, but the sector transfer is completed internally before the command phase is ended. The command phase is also terminated if the last sector on the track has been read or written, (as defined by the EOT parameter).

The 82072 activates INT to indicate the beginning of the result phase. The first reading of status from the 82072 resets the INT pin.

Result Phase

The generation of INT determines the beginning of the result phase. For each of the commands, a defined set of result bytes has to be read from the 82072 before the result phase is complete. (Refer to the Command Description section). These bytes of data must be read out for another command to start.

RQM and DIO must be equal to "1" and "1", before the result bytes may be read from the FIFO. After all the result bytes have been read, the RQM and DIO bits switch to "1" and "0" respectively, and the CM bit is cleared. This indicates that the 82072 is ready to accept the next command.

DATA RATE SELECT REGISTER (DSR)

The Data rate Select Register gives the user control over the read and write disk data. The user can select between an internal or external data separator, the data rate of the data separator and the delays for the write precompensation logic. The internal data separator requires no external control logic or analog components with its selectable data rates. SOFTWARE RESET and the POWERDOWN mode are invoked by setting the appropriate bit in this register.

When the processor writes into the DSR, the data is loaded after a delay to synchronize to an internal machine state. The processor must not perform successive writes into the DSR until this synchronization time (24 clock periods or 1 microsecond at 24 MHz) has elapsed. This register should be programmed before issuing a command that accesses the disk and uses values that this register controls. There is a 2 millisecond delay between writing the data rate select bits and having the PLL stabilize at the new frequency. This can be hidden by overlapping it in the

stepping and head load delays. There is no minimum delay between writing this register and accessing the FIFO and status registers.

A write to this register during data transfers will alter the contents of this register and the logic it controls. Data rate and precompensation values will change which may give undesirable results.

Changing the data rate also changes the timings of the drive control signals that are initialized with the SPECIFY command. To ensure that drive timings are not violated, the user should either choose values of drive timings such that the fastest data rate would not violate a drive timing specification or should follow a write to the data rate select bits with a new SPECIFY command. Refer to the SPECIFY command for more detail on these timings.

The Data rate Select Register (DSR) is a write only register. The functions of the DSR are indicated below:

SWR	PD	$\overline{\text{EPLL}}$	PRE-COMP			DRATESEL	
7	6	5	4	3	2	1	0

Bit 7—SWR; (SOFTWARE RESET)

When set to "1", enables software reset of the 82072. A software reset causes the hardware reset line to go active internally. Refer to Figure 5 for an illustration. This method ensures that software and hardware resets perform the same function. One function not reset is the DSR (since it contains the software reset bit). Since a hardware reset is being performed, the MSR will not be valid for up to 4 microseconds after the Software Reset is issued. The users software should wait that period before attempting to read the Main Status register. Software reset can also be cleared with a hardware reset.

In the case of software reset, the DSR retains the values loaded into it if DDRE is tied low. If DDRE is tied high, the precompensation and Data Rate Select bits will be reset to "00010". Upon hardware reset, the DSR will be configured based on DDRE setting. When DDRE is tied high, the DSR is configured as follows upon power up and reset:

SWR	PD	$\overline{\text{EPLL}}$	PRE-COMP			DRATESEL	
7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0

Default values: 250 Kbit/sec., Internal data separator, Precompensation = 125 ns.

When DDRE is tied low, the DSR retains the current data rate and precompensation values:

SWR	PD	$\overline{\text{EPLL}}$	PRE-COMP			DRATESEL	
7	6	5	4	3	2	1	0
0	0	0	X	X	X	X	X

Default values: X = current value, Internal data separator.

Bit 6—PD; (POWER DOWN)

When set to "1", the 82072 goes into its POWER DOWN mode. A SOFTWARE RESET is performed and held when entering POWER DOWN to ensure that no disk control pins are in the active state. The powerdown mode deactivates the internal clocks and shuts off the oscillator. Only writes to the DSR are allowed during powerdown and ALL input signals must be held in a valid state (either D.C. 1 or 0). The only exceptions are the data bus may be floated and an external crystal, which may remain running. There will be no increase in the ICC specification. To exit the Power down mode, either a hardware or software reset must be given. Once reset is given, there is a delay of 1 ms for the oscillator to stabilize at the correct frequency.

The 82072 preserves the contents of the PCN register during POWERDOWN. However, all status registers are cleared. Prior to issuing the POWERDOWN command, it is highly recommended that the user service all pending interrupts through the SENSE INTERRUPT STATUS command.

Bit 5—EPLL; (Enable PLL)

When set to "0", the internal PLL data separator supplies the data window input and the DW/DDRE pin operates in DDRE mode. The data rate of the internal PLL is determined by the DRATESEL bits. When set to "1", the internal data separator is disconnected from the serial interface controller and an external data separator is required to supply the DW signal. Refer to the section on the internal data separator for more detail.

Bits 2–4—PRECOMP; (Pre-compensation)

The write precompensation circuitry adjusts the write data pulse before it is sent to the drive on the WRDATA pin. A programmed compensation interval is added to or subtracted from the normal write pulse timing as a function of the data pattern. The CONFIGURE command is used to specify the track number that precompensation starts upon. If a CONFIGURE command is not issued, the 82072 defaults to beginning pre-compensation on Track 0.

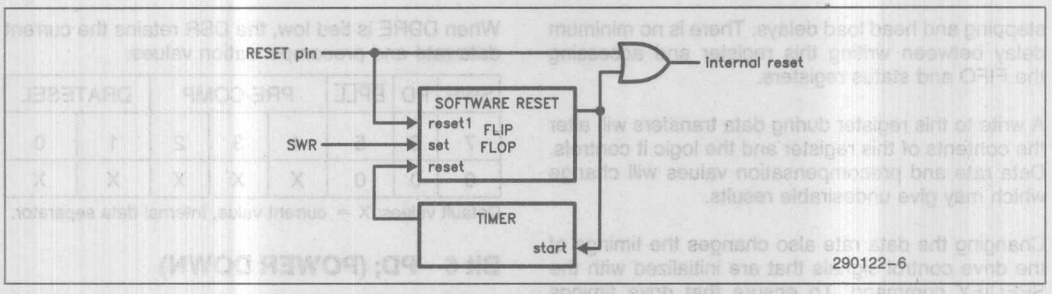


Figure 5. Reset Block Diagram

Table 5. Pre-Compensation Delays

PRECOMP 432	Pre-Compensation Delay
111	0.00 ns–DISABLED
001	41.67 ns
010	83.34 ns
011	125.00 ns
100	166.67 ns
101	208.33 ns
110	250.00 ns
000	DEFAULT

If the Data rate Select Register (DSR) is programmed with bits 2–4 set to zeroes, a default precompensation value is automatically chosen depending upon the data rate select bits (bits 0 and 1). The default values corresponding to the data rate select values are illustrated in Table 5A.

Table 5A. Default Pre-Compensation Delays

Data Rate	Pre-Compensation Delays
1 Mbps	41.67 ns
500 Kbps	125 ns
300 Kbps	125 ns
250 Kbps	125 ns

Bits 0, 1—DRATESEL; (Data Rate Select)

Programs both the read and write data rates. For single density (FM mode), data rates are one half the values stated for the double density (MFM) mode. MFM or FM selection is made in the disk read and write commands.

Table 4. Data Rates

DRATESEL 1	DRATESEL 0	DATA RATE MFM	DATA RATE FM
1	1	1 Mbps	Illegal
0	0	500 Kbps	250 Kbps
0	1	300 Kbps	150 Kbps
1	0	250 Kbps	125 Kbps

WRITE CLOCK

The basic "core" of the 8272A was carried over to the 82072 virtually unchanged. To keep the flexibility of the 8272A, where the user could vary data rates by changing the frequency on the WRCLK pin, and to minimize the external logic, a prescaler for the internal WRCLK was added. The prescaler divides down the 24 MHz clock to rates that are equivalent to those used with the 8272A. The user changes the prescaler value with the data rate selection bits in the DATA RATE SELECT register.

POLLING OF DRIVES

The 82072 defaults to polling enabled. Polling can be disabled by issuing a CONFIGURE command. When enabled, the 82072 polls the drives and looks for a change in the state of the RDY pin. Each of the drives is selected for a period of time and its RDY pin sampled. After a delay, the next drive is selected. This sequence occurs whenever the 82072 is waiting for a command or during SEEKs and RECALIBRATE, (but not IMPLIED SEEKs). The drives are assumed to be not ready after RESET and a "ready" value for each drive is saved in an internal register. An interrupt will be issued when a drive signals that it changed state from not ready to ready. This interrupt must be followed with a SENSE INTERRUPT STATUS command from the host. Each time that the drive changes its state from not ready to ready, an interrupt will again be issued.

The length of time that a drive is selected is dependent upon the selected data rate. At 1 Mbps, drives 0–2 are selected for approximately 20 microseconds, and drive 3 for about 440 microseconds. At 500 Kbps, the selected period is doubled for all drives. This period continues to scale with the data rate selection. The amount of time that the 82072 actually samples the RDY pin is about ½ microseconds after the drive select changes at 1 Mbps and also scales with the data rate. This value is equal for all drives. The remaining time the drive is selected, changes on RDY will not be sampled.

NOTE:
Polling is disabled when an Implied Seek is in progress.

TERMINAL COUNT (TC)

TC completes the current disk data transfer and then puts the 82072 into the result phase.

DRQ or RQM/INT in the Non-DMA mode will go inactive immediately after TC is received. There will be no more requests for data transfers. In a write command, all remaining bytes in the FIFO are read out by the 82072. The remaining space in the disk sector will be filled with "00"s. For read's, the remaining data bytes are "dumped" internally and CRC is checked. An overrun/underrun error will not be reported in the result phase. The FIFO will be reset upon entering the result phase. This ensures that any remaining data bytes sent by the host do not interfere with the result bytes. Care should be taken not to leave TC active in the result phase as it will cause the result phase to be terminated. If necessary, TC can be negated by the active edge of INT such that the removal is within 500 ns.

It should be noted that if TC is asserted without an accompanying $\overline{DACK}$ or $\overline{CS}$ and $\overline{RD}/\overline{WR}$ (FIFO transfer), then DRQ remains high until the pending FIFO transaction is performed. However, the receipt of TC is sufficient to internally inhibit further data requests.

The 82072 supports terminal count implicitly through the underrun/overrun and end-of-track (EOT) functions. For full sector transfer, the EOT parameter can define the last sector to be transferred in a single or multisector transfer. If the last sector to be transferred is a partial sector, the host can stop transferring the data in mid-sector and the 82072 will complete the transfer in the same way as if a hardware TC was received. The only difference between these implicit functions and TC is that they return "abnormal termination" result status. Such status indications can be ignored if they were expected.

It is recommended that the TC be issued with the last byte transferred, or shortly thereafter. The user must issue a TC before GPL byte times of the request for the last byte of the sector, or the 82072 will continue on to the next sector. This may cause an error condition such as reading past end-of-track, underrun/overrun of the FIFO, or a sector not found.

Terminal count will always get the 82072 out of the execution phase which would be caused by the host failing to perform the correct action.

DMA TRANSFERS

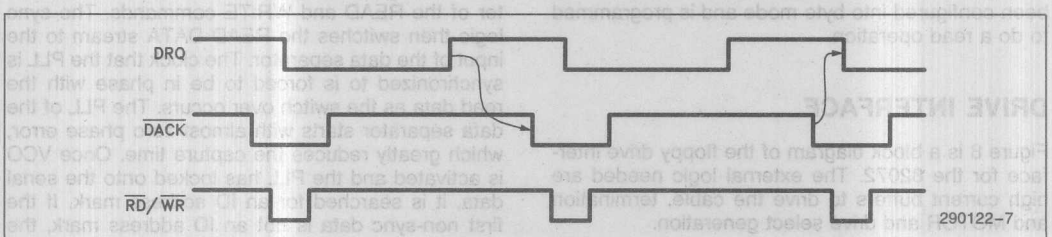
DMA transfers are initiated by the 82072 when it activates its DRQ (DMA Request) pin. When ready, the DMA controller responds by activating $\overline{DACK}$ (DMA Acknowledge). If the DMA transfer mode is selected in the 82072 (with the SPECIFY command), DRQ must be used as the RQM bit signal since RQM is not activated. $\overline{DACK}$ or CS = 0 and A0 = 1 can be used to respond to the request.

If the FIFO is enabled, DRQ will go active for a transfer and stay active until the FIFO is filled or cleared (write and read respectively).

If the FIFO is disabled (default value) by setting the EFIFO bit in the CONFIGURE command, each byte to be transferred will request a cycle. Figure 6 is an example of the handshaking that occurs.

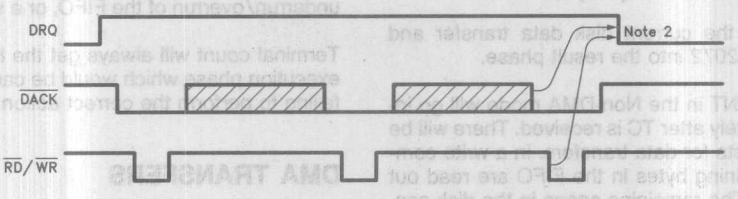
The 82072 can be given pulsed $\overline{DACK}$'s (which is compatible with the iAPX186 and most two cycle DMA controllers) or $\overline{DACK}$'s that stay active until the transfer is complete (compatible with the Intel 8237A controller). A pulsed $\overline{DACK}$ provides more time for the DMA controller to determine if DRQ should be removed (since $\overline{DACK}$ goes active before RD). If $\overline{DACK}$ is not pulsed, then the active going edge of RD is used for this decision. The edge of RD may be too late for DRQ to be removed and prevent an unwanted DMA cycle.

There is a small delay after $\overline{DACK}$ (or RD) goes active at the 82072 before DRQ is removed. This may not satisfy the DMA controller specification ($\overline{DACK}$ active to DRQ inactive) to prevent an unwanted DMA cycle. It is the users responsibility to ensure that an unwanted cycle does not happen due to DRQ not going inactive quickly enough. It may be



NOTE:
The falling edge of $\overline{DACK}$ asynchronously removes DRQ.

Figure 6. Disabled FIFO DMA Cycles



NOTES:
1. DACK can be either active the entire period or can be active for each cycle.
2. Either edge will remove DRQ, but DACK provides more setup time for the 82072 and the DMA controller.

Figure 7. DMA Transfers with the FIFO Enabled

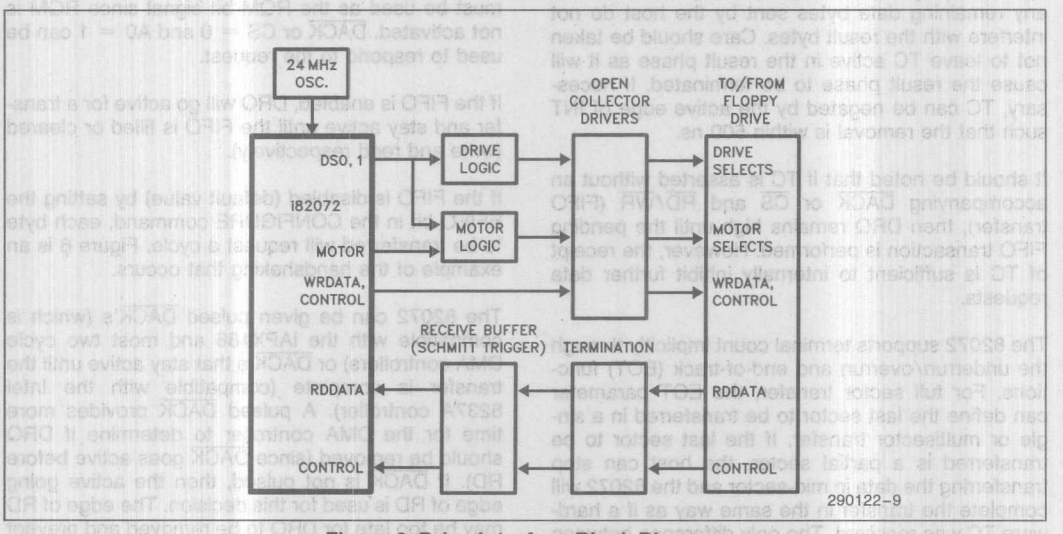


Figure 8. Drive Interface Block Diagram

necessary for the design to generate a pulsed DACK, one that is earlier than the one supplied by the DMA controller or add wait states.

Note that if the DMA controller (i.e., 8237A) is programmed to function in verify mode, a pseudo read is performed by the 82072 based only on DACK. This mode is only available when the 82072 has been configured into byte mode and is programmed to do a read operation.

DRIVE INTERFACE

Figure 8 is a block diagram of the floppy drive interface for the 82072. The external logic needed are high current buffers to drive the cable, termination and MOTOR and drive select generation.

When a read operation begins, the sync detect logic samples the read data stream until it finds a valid sync field, which is a pattern of eight contiguous zero bit cells. This is found with an internal one-shot. It assumes that this is a synchronization field (Refer to the FORMAT command for detail) and will switch the VCO from the reference clock to the data stream after waiting the delay specified in the GAP parameter of the READ and WRITE commands. The sync logic then switches the READ DATA stream to the input of the data separator. The clock that the PLL is synchronized to is forced to be in phase with the read data as the switch over occurs. The PLL of the data separator starts with almost zero phase error, which greatly reduces the capture time. Once VCO is activated and the PLL has locked onto the serial data, it is searched for an ID address mark. If the first non-sync data is not an ID address mark, the VCO line is deactivated and the search for a sync.

field begins again with the data separator waiting for the one-shot to find another sync. field. If the address mark was good, the ID field is examined for the correct parameters and CRC is checked. When the 82072 is not looking at the disk data, it remains synchronized to the programmed data rate. This method ensures that the PLL does not lock onto harmonics. It also allows a faster lock-up time when disk data is fed into the PLL.

DATA SEPARATOR

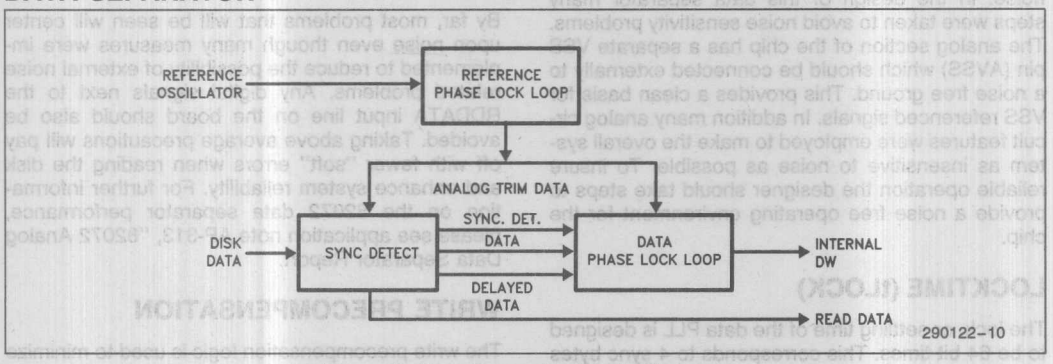


Figure 9. Data Separator Block Diagram

The function of the data separator is to lock onto the incoming serial read data. When lock is achieved the serial front end logic of the chip is provided with a clock which is synchronized to the read data. The synchronized clock, called Data Window, is used to internally sample the serial data. One state of Data Window is used to sample the data portion of the bit cell, and the alternate state samples the clock portion. Serial to parallel conversion logic separates the read data into clock and data bytes.

To support reliable disk reads the data separator must track fluctuations in the disk data frequency. Frequency errors primarily arise from two sources: drive rotation speed variation and instantaneous speed variation (ISV). A second condition, and one that opposes the ability to track frequency shifts is the response to bit jitter.

The internal data separator consists of two analog phase lock loops (PLLs) as shown in Figure 9. The two PLLs are referred to as the reference PLL and the data PLL. The reference PLL (the master PLL) is used to bias the data PLL (the slave PLL). The reference PLL adjusts the data PLL's operating point as a function of process, junction temperature and supply voltage. Using this architecture it was possible to eliminate the need for external trim components.

Figure 10 shows the data PLL. The reference PLL has control over the loop gain by its influence on the

PHASE LOCK LOOP OVERVIEW

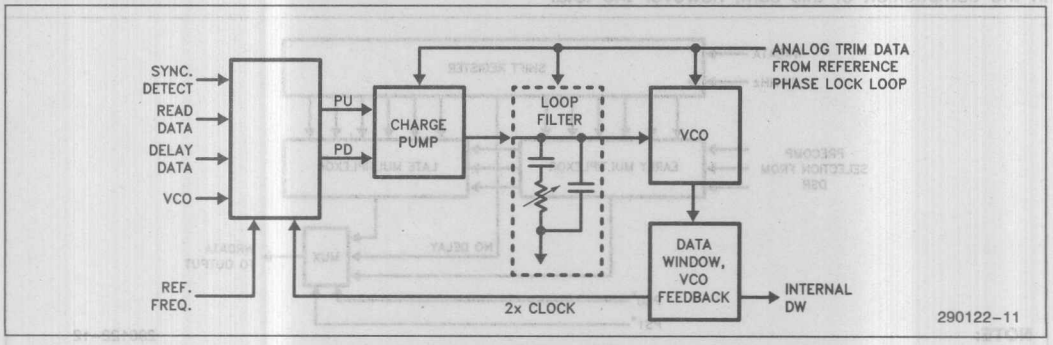


Figure 10. Data PLL

charge pump and the VCO. In addition the reference PLL controls the loop filter time constant. As a result the closed loop transfer function of the data PLL is controlled, and immune, to the first order, to environmental factors, and process variation.

Systems of this type are often very sensitive to noise. In the design of this data separator many steps were taken to avoid noise sensitivity problems. The analog section of the chip has a separate VSS pin (AVSS) which should be connected externally to a noise free ground. This provides a clean basis for VSS referenced signals. In addition many analog circuit features were employed to make the overall system as insensitive to noise as possible. To insure reliable operation the designer should take steps to provide a noise free operating environment for the chip.

LOCKTIME (tLOCK)

The lock, or settling time of the data PLL is designed to be 64 bit times. This corresponds to 4 sync bytes in the FM mode and 8 sync bytes in the MFM mode. This value assumes that the sync field jitter is 5% the bit cell or less. This level of jitter should be easily achieved for a constant bit pattern, since intersymbol interference should be equal, thus nearly eliminating random bit shifting.

CAPTURE RANGE (tRANGE)

The 82072 data separator is designed to lock onto RDDATA with a maximum frequency error of approximately ±5.5%. The graphs in Figures 17 and 18 in the Data Separator Characteristics Section illustrate the capture range around the center frequency. This value is made up of two components: drive motor speed error and ISV. Frequency is a factor which may determine the maximum level of the ISV component. In general, as frequency increases the allowed magnitude of the ISV component will decrease. The designer has a certain level of flexibility in the composition of this sum, however the total must never exceed ±5.5%. In order to insure the reliability of dual drive systems, designers should require drives with a motor speed variation of ±2.75%, resulting in a worse case of ±5.5% when the drives are switched.

SYSTEM CONSIDERATIONS

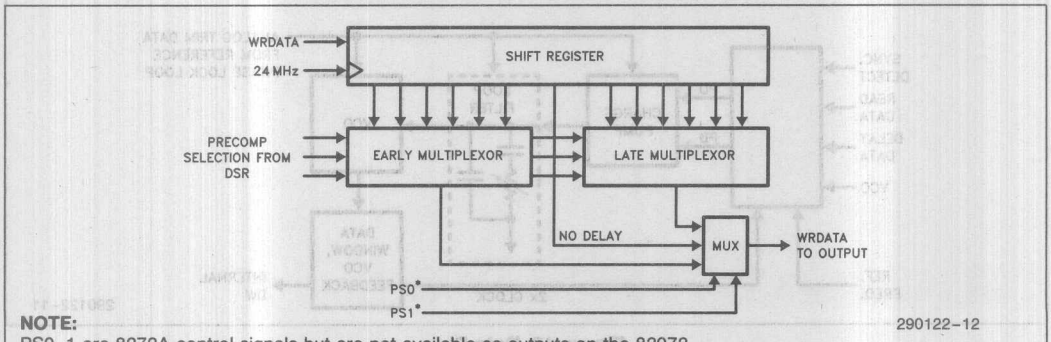
By far, most problems that will be seen will center upon noise even though many measures were implemented to reduce the possibility of external noise related problems. Any digital signals next to the RDDATA input line on the board should also be avoided. Taking above average precautions will pay off with fewer "soft" errors when reading the disk and enhance system reliability. For further information on the 82072 data separator performance, please see application note AP-313, "82072 Analog Data Separator Report."

WRITE PRECOMPENSATION

The write precompensation logic is used to minimize bit shifts in the RDDATA stream from the disk drive. The shifting of bits is a known phenomena of magnetic media and is dependent upon the disk media AND the floppy drive.

The 82072 monitors the bit stream that is being sent to the drive. The data patterns that require precompensation are well known. Depending upon the pattern, the bit is shifted either early or late (or not at all) relative to the surrounding bits. Figure 11 is a block diagram of the internal circuit.

The top block is a 13 bit shift register with the no delay tap being in the center. This allows 6 levels of early and late shifting with respect to nominal. The shift register is clocked at the main clock rate (24 MHz). The output is fed into 2 multiplexors—one for early and one for late. A final stage of multiplexors combines the early, late and normal data stream back into one which is the WRDATA output.



NOTE:
PS0, 1 are 8272A control signals but are not available as outputs on the 82072.

Figure 11. Precompensation Block Diagram

COMMAND SET

Table 5B. 82072 Command Set

Phase	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W					C				Sector ID information prior to Command execution
	W					H				
	W					R				
	W					N				
	W					EOT				
	W					GPL				
	W					DTL				
Execution										Data transfer between the FDD and main-system
Result	R					ST 0				Status information after Command execution
	R					ST 1				
	R					ST 2				
	R					C				Sector ID information after Command execution
	R					H				
	R					R				
	R					N				
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W					C				Sector ID information prior to Command execution
	W					H				
	W					R				
	W					N				
	W					EOT				
	W					GPL				
	W					DTL				
Execution										Data transfer between the FDD and main-system
Result	R					ST 0				Status information after Command execution
	R					ST 1				
	R					ST 2				
	R					C				Sector ID information after Command execution
	R					H				
	R					R				
	R					N				
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W					C				Sector ID information prior to Command execution
	W					H				
	W					R				
	W					N				
	W					EOT				
	W					GPL				
	W					DTL				
Execution										Data transfer between the main-system and FDD
Result	R					ST 0				Status information after Command execution
	R					ST 1				
	R					ST 2				
	R					C				Sector ID information after Command execution
	R					H				
	R					R				
	R					N				

Table 5B. 82072 Command Set (Continued)

Phase	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W					C				Sector ID information prior to Command execution
	W					H				
	W					R				
	W					N				
	W					EOT				
	W					GPL				
	W					DTL				
Execution										Data transfer between the FDD and main-system
Result	R					ST 0				Status information after Command execution
	R					ST 1				
	R					ST 2				
	R					C				
	R					H				Sector ID information after Command execution
	R					R				
	R					N				
READ TRACK										
Command	W	0	MFM	0	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W					C				Sector ID information prior to Command execution
	W					H				
	W					R				
	W					N				
	W					EOT				
	W					GPL				
	W					DTL				
Execution										Data transfer between the FDD and main-system. FDC reads all of cylinders contents from index hole to EOT
Result	R					ST 0				Status information after Command execution
	R					ST 1				
	R					ST 2				
	R					C				
	R					H				Sector ID information after Command execution
	R					R				
	R					N				
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Commands
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the Cylinder is stored in Data Register
Result	R					ST 0				Status information after Command execution
	R					ST 1				
	R					ST 2				
	R					C				
	R					H				Disk status after the Command has completed.
	R					R				
	R					N				

Table 5B. 82072 Command Set (Continued)

Phase	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
FORMAT TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	←			N			→		Bytes/Sector
	W	←			SC			→		Sectors/Cylinder
	W	←			GPL			→		Gap 3
	W	←			D			→		Filler Byte
For Each Sector Repeat:	W	←			C			→		Input Sector Parameters
	W	←			H			→		
	W	←			R			→		
	W	←			N			→		
Execution										FDC formats an entire cylinder
Result	R	←			ST 0			→		Status information after Command execution
	R	←			ST 1			→		
	R	←			ST 2			→		
	R	←			Undefined			→		
	R	←			Undefined			→		
	R	←			Undefined			→		
	R	←			Undefined			→		
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head retracted to Track 0 Interrupt
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R	←			ST 0			→		Status information at the end of each seek operation about the FDC
	R	←			PCN			→		
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	← SRT →					← HUT →			
	W	← HLT →							ND	
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	MOT	0	0	0	0	HDS	DS1	DS0	
Result	R	←			ST 3			→		Status information about FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	←			NCN			→		
Execution										Head is positioned over proper Cylinder on Diskette
CONFIGURE										
Command	W	0	0	0	1	0	0	1	1	Configure Information
	W	HSDA	←	MOFF	→		←	MON	→	
	W	0	EIS	EFIFO	POLL		←	FIFOTHR	→	
	W	←			PRETRK			→		
MOTOR ON/OFF										
Command	W	On/Off	DS1	DS0	0	1	0	1	1	Command Code
RELATIVE SEEK										
Command	W	1	DIR	0	0	1	1	1	1	
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	←			RCN			→		

Table 5B. 82072 Command Set (Continued)

Phase	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Remarks
DUMPREG										
Command	W	0	0	0	0	1	1	1	0	*Note
Execution										Registers placed in FIFO
Result	R	———			PCN-Drive 0				———	
	R	———			PCN-Drive 1				———	
	R	———			PCN-Drive 2				———	
	R	———			PCN-Drive 3				———	
	R	<———		SRT	———>	<———		HUT	———>	
	R	<———			HLT	———			>	ND
	R	<———			SC/EOT	———			———	>
	R	HSDA <———		MOFF	———>	<———		MON	———>	See Note
	R	0	EIS	EFIFO	POLL	<———		FIFOTHR	———>	
	R	<———			PRETRK	———			———	>
INVALID										
Command	W	———			Invalid Codes	———			———	Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R	———			ST 0	———			———	ST 0 = 80

(16)

SC is returned if the last command that was issued was the FORMAT command. EOT is returned if the last command was a READ or WRITE.

NOTES:
1. Symbols used in this table are described below.
2. A_0 = 1 for all operations.

PARAMETER ABBREVIATIONS:

Symbol	Description
C	Cylinder address. The currently selected cylinder address, 0 to 255.
D	Data pattern. The pattern to be written in each sector data field during formatting.
	DS0, DS1 Disk Drive Select.

DS1	DS0	
0	0	drive 0
0	1	drive 1
1	0	drive 2
1	1	drive 3

DTL	Special sector size. By setting N to zero (00), DTL may be used to control the number of bytes transferred in disk read/write commands. The sector size (N = 0) is set to 128. If the actual sector (on the diskette) is larger than DTL, the remainder of the actual sector is read but is not passed to the host during read commands; during write commands, the remainder of the actual sector is written with all zero bytes. The CRC check code is calculated with the actual sector. When N is not zero, DTL has no meaning and should be set to FF HEX.
EOT	End of track. The final sector number of the current track.

Symbol	Description
GPL	Gap length. The gap 3 size. (Gap 3 is the space between sectors excluding the VCO synchronization field).
H/HDS	Head address. Selected head: 0 or 1 (disk side 0 or 1) as encoded in the sector ID field.
HLT	Head load time. The time interval that 82072 waits after loading the head and before initiating a read or write operation. Refer to the SPECIFY command for actual delays.
HUT	Head unload time. The time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Refer to the SPECIFY command for actual delays.
MFM	MFM/FM mode selector. A one selects the double density (MFM) mode. A zero selects single density (FM) mode.
MT	Multi-track selector. When set, this flag selects the multi-track operating mode. In this mode, the 82072 treats a complete cylinder, under head 0 and 1, as a single track. The 82072 operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set, a multitrack read or write operation will automatically continue to the first sector

Symbol	Description
	under head 1 when the 82072 finishes operating on the last sector under head 0.
N	Sector size code. This specifies the number of bytes in a sector. If this parameter is "00", then the sector size is 128 bytes. The number of bytes transferred is determined by the DTL parameter. Otherwise the sector size is (2 raised to the "N'th" power) times 128. All values up to "07" hex are allowable. "07"h would equal a sector size of 16k. It is the users responsibility to not select combinations that are not possible with the drive.

N	Sector Size
00	128 bytes
01	256 bytes
02	512 bytes
03	1024 bytes
..	...
07	16 Kbytes

Symbol	Description
NCN	New cylinder number. The desired cylinder number.
ND	Non-DMA mode flag. When set to 1, indicates that the 82072 is to operate in the non-DMA mode. In this mode, the host is interrupted for each data transfer. When set to 0, the 82072 operates in DMA mode, interfacing to a DMA controller by means of the DRQ and DACK# signals.
PCN	Present cylinder number. The current position of the head at the completion of SENSE INTERRUPT STATUS command.
R	Sector address. The sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of the first sector to be read or written.
SC	Number of sectors per track. The number of sectors per track to be initialized by the FORMAT command.
SK	Skip flag. When set to 1, sectors containing a deleted data address mark will automatically be skipped during the execution of READ DATA. If READ DELETED is executed, only sectors with a deleted address mark will be accessed. When set to "0", the sector is read or written the same as the read and write commands.
SRT	Step rate interval. The time interval between step pulses issued by the 82072. Programmable from 0.5 to 8 milliseconds, in increments of 0.5 ms at the 1 Mbit data rate. Refer to the SPECIFY command for actual delays.
ST0 ST1 ST2 ST3	Status register 0–3. Registers within the 82072 that store status information after a command has been executed. This status information is available to the host during the result phase after command execution.

Commands can be written whenever the 82072 is in the command phase. The command set falls into two categories as is shown in Table 6. The encoding of the bits used in this description can be found in the Parameter Abbreviation section which follows this section.

Each command has a unique set of parameters that are needed and that are returned as status. The 82072 checks to see that the first byte received while in the command phase is a valid command. If valid, there will be no change in the status bits in the MSR. If the value was not a valid command, an interrupt is issued. The user would issue the SENSE INTERRUPT STATUS command which would return an invalid command error condition.

DATA TRANSFER COMMANDS

The READ DATA, READ DELETED DATA, WRITE DATA, WRITE DELETED DATA and READ TRACK require and return the same parameter and status bytes respectively. The only differences being the coding of bits 0–4 in the first byte sent to the 82072.

Table 6. Command Summary

Data Transfer	Control	
Read Data	Recalibrate	Configure *
Read Deleted Data	Sense Interrupt Status	Motor On/Off *
Write Data	Specify	Relative Seek*
Write Deleted Data	Sense Drive Status	Dumpreg *
Read Track	Seek	
Format Track	Read ID	

***NOTE:**
These commands are not present in the 8272A.

Table 7. Data Transfer Command Summary

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	MT	MFM	SK	0	X	X	X	X	Command Code
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	—	—	—	—	C	—	—	—	
	W	—	—	—	—	H	—	—	—	
	W	—	—	—	—	R	—	—	—	
	W	—	—	—	—	N	—	—	—	
	W	—	—	—	—	EOT	—	—	—	
	W	—	—	—	—	GPL	—	—	—	
	W	—	—	—	—	DTL	—	—	—	
Execution										Data Transfer Occurs.
Result	R	—	—	—	—	ST0	—	—	—	Status After
	R	—	—	—	—	ST1	—	—	—	Command Has
	R	—	—	—	—	ST2	—	—	—	Completed.
	R	—	—	—	—	C	—	—	—	Disk Status
	R	—	—	—	—	H	—	—	—	After The
	R	—	—	—	—	R	—	—	—	Command Has
	R	—	—	—	—	N	—	—	—	Completed.

Table 8. Command Encoding

Command	D3	D2	D1	D0
Read Data	0	1	1	0
Read Deleted Data	1	1	0	0
Write Data	0	1	0	1
Write Deleted Data	1	0	0	1
Read Track	0	0	1	0

IMPLIED SEEK FOR READ AND WRITE COMMANDS

The 82072 will execute an implied seek for any disk transfer command that specifies the cylinder number. The implied seek execution is transparent to the user. This means that the user would issue a READ DATA command, send all of the parameter bytes, see step pulses issued and start reading data.

After the parameter bytes are received, a comparison is made between PCN and C. If C does not equal PCN and the Implied Seek bit in the CONFIGURE command was set, then a SEEK command is issued. This would be indicated by the DRIVE BUSY bits in the Main Status Register being set. If the stepping completes successfully, DRIVE BUSY in the MSR will go inactive and then the disk transfer command starts. If the SEEK failed, the status bytes of the command are sent to the FIFO. ST0 contains the cause of the error and C would contain the cylinder number on which the SEEK failed.

READ DATA COMMAND

A set of nine (9) bytes is required to place the 82072 into the Read Data Mode. After the READ DATA command has been issued, the 82072 loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the SPECIFY command), and begins reading ID Address Marks and ID fields. When the sector address read off the diskette matches with the sector address specified in the command, the 82072 reads the sector's data field and transfers the data to the FIFO.

After completion of the read operation from the current sector, the sector address is incremented by one, and the data from the next logical sector is read and ouput via the FIFO. This continuous read function is called "Multi-Sector Read Operation". Upon receipt of TC, or an implied TC (FIFO overrun/underrun), the 82072 stops sending data, but will continue to read data from the current sector, check the CRC bytes, and at the end of the sector terminate the READ DATA Command.

N determines the number of bytes per sector (See Table 8A Below). If N is set to zero, the sector size is set to 128. The DTL value determines the number of bytes to be transferred. If DTL is less than 128, the 82072 transfers the specified number of bytes to the host. For reads, it continues to read the entire 128 byte sector and checks for CRC errors. For writes it completes the 128 byte sector by filling in zeroes. If N is not set to 00 Hex, DTL should be set to FF Hex, and has no impact on the number of bytes transferred.

Table 8A. Sector Sizes

N	Sector Size
00	128 bytes
01	256 bytes
02	512 bytes
03	1024 bytes
. .	. . .
07	16 Kbytes

The amount of data which can be handled with a single command to the 82072 depends upon MT (multi-track) and N (Number of bytes/sector).

MT	N	Max. Transfer Capacity	Final Sector Read from Disk
0	1	256 × 26 = 6,656	26 at side 0 or 1
1	1	256 × 52 = 13,312	26 at side 1
0	2	512 × 15 = 7,680	15 at side 0 or 1
1	2	512 × 30 = 15,360	15 at side 1
0	3	1024 × 8 = 8,192	8 at side 0 or 1
1	3	1024 × 16 = 16,384	16 at side 1

The Multi-Track function (MT) allows the 82072 to read data from both sides of the diskette. For a particular cylinder, data will be transferred starting at Sector 1, Side 0 and completing at the last sector of the same track at Side 1.

At the completion of the READ DATA Command, the head is not unloaded until after the Head Unload Time Interval (specified in the SPECIFY command) has elapsed. If the host issues another command before the head unloads then the head settling time may be saved between subsequent reads.

If the 82072 detects a pulse on the IDX pin twice without finding the specified sector (meaning that the diskette's index hole passes through index detect logic in the drive twice), the 82072 sets the IC code in Status Register 0 to "01" (Abnormal termination), and sets the ND bit in Status Register 1 to "1" indicating a sector not found, and terminates the READ DATA Command.

After reading the ID and Data Fields in each sector, the 82072 checks the CRC bytes. If a CRC error occurs in the ID or data field, the 82072 sets the IC code in Status Register 0 to "01" (Abnormal termination), sets the DE bit flag in Status Register 1 to "1", sets the DD bit in Status Register 2 to "1" if CRC is incorrect in the ID field, and terminates the READ DATA Command.

Table 8B below describes the affect of the SK bit on the READ DATA command execution and results.

Table 8B. Skip Bit vs. READ DATA Command

SK Bit Value	Data Address Mark Type Encountered	Results		
		Sector Read?	CM Bit of ST2 Set?	Description of Results
0	Normal Data	Yes	No	Normal Termination.
0	Deleted Data	Yes	Yes	Address Not Incremented. Next Sector Not Searched For.
1	Normal Data	Yes	No	Normal Termination.
1	Deleted Data	No	Yes	Normal Termination Sector Not Read ("Skipped").

Except where noted in Table 8B, the C or R value of the sector address is automatically incremented (see Table 8D).

WRITE DATA

After the WRITE DATA command has been issued, the 82072 loads the head (if it is in the unloaded state), waits the specified head load time if unloaded (defined in the SPECIFY command), and begins reading ID Fields. When the sector address read from the diskette matches the sector address specified in the command, the 82072 reads the data from the host via the FIFO, and writes it to the sector's data field.

After writing data into the current sector, the 82072 computes the CRC value and writes it into the CRC field at the end of the sector transfer. The Sector Number stored in "R" is incremented by one, and the 82072 continues writing to the next data field. The 82072 continues this "Multi-Sector Write Operation". Upon receipt of a terminal count signal or if a FIFO over/under run occurs while a data field is being written, then the remainder of the data field is filled with zeros.

The 82072 reads the ID field of each sector and checks the CRC bytes. If it detects a CRC error in one of the ID Fields, it sets the IC code in Status Register 0 to "01" (Abnormal termination), sets the DE bit of Status Register 1 to "1", and terminates the WRITE DATA command.

The WRITE DATA command operates in much the same manner as the READ DATA command. The following items are the same. Please refer to the READ DATA Command for details:

- Transfer Capacity
- EN (End of Cylinder) bit
- ND (No Data) bit
- Head Load, Unload Time Interval
- ID information when the host terminates the command.
- Definition of DTL when N = 0 and when N does not = 0.

READ DELETED DATA

This command is the same as the READ DATA command, only it operates on sectors that contain a Deleted Data Address Mark at the beginning of a Data Field.

Table 8C describes the affect of the SK bit on the READ DELETED DATA command execution and results.

Table 8C. Skip Bit vs READ DELETED DATA Command

SK Bit Value	Data Address Mark Type Encountered	Results		
		Sector Read?	CM Bit of ST2 Set?	Description of Results
0	Normal Data	Yes	Yes	Address Not Incremented. Next Sector Not Searched For.
0	Deleted Data	Yes	No	Normal Termination.
1	Normal Data	No	Yes	Normal Termination Sector Not Read ("Skipped").
1	Deleted Data	Yes	No	Normal Termination.

Except where noted in Table 8C above, the C or R value of the sector address is automatically incremented (See Table 8D).

WRITE DELETED DATA

This command is almost the same as the WRITE DATA command except that a Deleted Data Address Mark is written at the beginning of the Data Field instead of the normal Data Address Mark. This command is typically used to mark a bad sector containing an error on the floppy disk.

READ TRACK

This command is similar to the READ DATA command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering a pulse on the IDX pin, the 82072 starts to read all data fields on the track as continuous blocks of data without regard to logical sector numbers. If the 82072 finds an error in the ID or DATA CRC check bytes, it continues to read data from the track and sets the appropriate error bits at the end of the command. The 82072 compares the ID information read from each sector with the specified value in the command, and sets the ND flag of Status Register 1 to a "1" if there is no comparison. Multi-track or skip operations are not allowed with this command. The SK bit should always be set to "0".

This command terminates when the EOT specified number of sectors have been read. If the 82072 does not find an ID Address Mark on the diskette after the second occurrence of a pulse on the IDX pin, then it sets the IC code in Status Register 0 to "01" (Abnormal termination), sets the MA bit in Status Register 1 to "1", and terminates the command.

REGISTER INFORMATION FOR HOST TERMINATION

If the host terminates a read or write operation in the 82072, then the ID information in the result phase is dependent upon the state of the MT bit and EOT byte.

Table 8D. Result Phase Table

MT	Head	Final Sector Transferred to Host	ID Information at Result Phase			
			C	H	R	N
0	0	Less than EOT	NC	NC	R+1	NC
		Equal to EOT	C+1	NC	01	NC
	1	Less than EOT	NC	NC	R+1	NC
		Equal to EOT	C+1	NC	01	NC
1	0	Less than EOT	NC	NC	R+1	NC
		Equal to EOT	NC	LSB	01	NC
	1	Less than EOT	NC	NC	R+1	NC
		Equal to EOT	C+1	LSB	01	NC

NC: no change, the same value as the one at the beginning of command execution.
LSB: least significant bit, the LSB of H is complemented.

FORMAT COMMAND

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	MFM	0	0	1	1	0	1	Command Code
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	—	—	—	—	N	—	—	—	
	W	—	—	—	—	SC	—	—	—	
	W	—	—	—	—	GPL	—	—	—	
	W	—	—	—	—	D	—	—	—	
For Each Sector Repeat:	W	—	—	—	—	C	—	—	—	Input Sector Parameter
	W	—	—	—	—	H	—	—	—	
	W	—	—	—	—	R	—	—	—	
	W	—	—	—	—	N	—	—	—	
Execution										Data Transfer Occurs
Result										
	R	—	—	—	—	ST0	—	—	—	Status After Command Has Completed.
	R	—	—	—	—	ST1	—	—	—	
	R	—	—	—	—	ST2	—	—	—	
	R	—	—	—	—	Undefined	—	—	—	
	R	—	—	—	—	Undefined	—	—	—	
	R	—	—	—	—	Undefined	—	—	—	
	R	—	—	—	—	Undefined	—	—	—	

The FORMAT command allows an entire track to be formatted. After a pulse from the IDX pin is detected, the 82072 starts writing data on the disk including Gaps, Address Marks, ID Fields and Data Fields, per the IBM System 34 or 3740 format (MFM or FM respectively). The particular values that will be written to the gap and data field are controlled by the values programmed into N, SC, GPL, and D which are specified by the host during the command phase. The data field of the sector is filled with the data byte specified by D. The ID Field for each sector is supplied by the host; that is, four data bytes per sector are needed by the 82072 for C, H, R, and N (cylinder, head, sector number and sector size respectively).

After formatting each sector, the host must send new values for C, H, R and N to the 82072 for the next sector on the track. The R value (sector number) is the only value that must be changed by the host after each sector

is formatted. This allows the disk to be formatted with nonsequential sector addresses (interleaving). This incrementing and formatting continues for the whole track until the 82072 encounters a pulse on the IDX pin again and it terminates the command.

Table 9 contains typical values for gap fields which are dependent upon the size of the sector and the number of sectors on each track. Actual values can vary due to drive electronics.

Table 9. Typical Values for Formatting

		Sector Size	N	SC	GPL1	GPL2
8″ Drives	FM	128	0	1A	07	1B
		256	1	0F	0E	2A
		512	2	08	1B	3A
		1024	3	04	47	8A
		2048	4	02	C8	FF
		4096	5	01	C8	FF
	MFM	256	1	1A	0E	36
		512	2	0F	1B	54
		1024	3	08	35	74
		2048	4	04	99	FF
		4096	5	02	C8	FF
5.25″ Drives	FM	128	00	12	07	09
		128	00	10	10	19
		512	02	08	18	30
		1024	03	04	46	87
		2048	04	02	C8	FF
		4096	05	01	C8	FF
		...	...			
	MFM	256	01	12	0A	0C
		256	01	10	20	32
		512	02	08	2A	50
		1024	03	04	80	F0
		2048	04	02	C8	FF
		4096	05	01	C8	FF
		...	...			
3.5″ Drives	FM	128	0	0F	07	1B
		256	1	09	0F	2A
		512	2	05	1B	3A
	MFM	256	1	0F	0E	36
		512	2	09	1B	54
		1024	3	05	35	74

GPL1 = suggested GPL values in read and write commands to avoid splice point between data field and ID field of contiguous sections.
GPL2 = suggested GPL value in FORMAT TRACK command.

NOTE:
All values are in Hex.

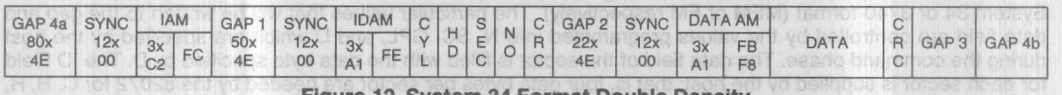


Figure 12. System 34 Format Double Density

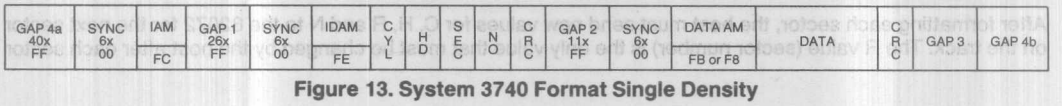


Figure 13. System 3740 Format Single Density

CONTROL COMMANDS

Each of the control commands has a unique set of parameters that are sent and returned. Control commands differ in that they do not have a data transfer in the execution phase.

CONTROL COMMANDS WITH INTERRUPTS

READ ID COMMAND

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	MFM	0	0	1	0	1	0	Command Code
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										1ST ID Field Located.
Result	R					ST0				Status After
	R					ST1				Command Has
	R					ST2				Completed.
	R					C				Disk Status
	R					H				After The
	R					R				Command Has
	R					N				Completed.

The READ ID command is used to find the present position of the recording heads. The 82072 stores the values from the first ID Field it is able to read into its registers. If the 82072 does not find an ID Address Mark on the diskette after the second occurrence of a pulse on the IDX pin, it then sets the IC code in Status Register 0 to "01" (Abnormal termination), sets the MA bit in Status Register 1 to "1", and terminates the command.

The following commands will generate an interrupt upon completion. They do not return any result bytes. It is highly recommended that control commands be followed by the SENSE INTERRUPT STATUS command. Otherwise, valuable interrupt status information will be lost.

RECALIBRATE COMMAND

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	0	0	0	0	1	1	1	Command Code
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head Retracts To Track 0.
										Interrupt

This command causes the read/write head within the FDD to retract to the track 0 position. The 82072 clears the contents of the PCN counter, and checks the status of the TRK0 pin from the FDD. As long as the TRK0 pin is low, the DIR pin remains 0 and step pulses are issued. When the TRK0 pin goes high, the SE bit in Status Register 0 is set to "1", and the command is terminated. If the TRK0 pin is still low after 255 step pulses have been issued, the 82072 sets the SE and the EC bits of Status Register 0 to "1", and terminates the command. Disks capable of handling more than 256 tracks per side may require more than one RECALIBRATE command to return the head back to physical Track 0.

The RECALIBRATE command does not have a result phase. SENSE INTERRUPT STATUS command must be issued after the RECALIBRATE command to effectively terminate it and to provide verification of the head position (PCN). During the command phase of the recalibrate operation, the 82072 is in the BUSY state, but during the execution phase it is in a NON BUSY state. At this time another RECALIBRATE command may be issued, and in this manner, parallel RECALIBRATE operations may be done on up to 4 drives at once.

Upon power up, the software must issue a RECALIBRATE command to properly initialize all drives and the controller.

SEEK COMMAND

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	0	0	0	1	1	1	1	Command Code
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	—	—	—	—	NCN	—	—	—	
Execution										Head is Moved to Specified Track.

The read/write head within the FDD is moved from track to track under the control of the SEEK Command. The 82072 compares the PCN which is the current head position with the NCN and performs the following operation if there is a difference:

—PCN < NCN: Direction signal to FDD set to "1" (step in), and issues step pulses.

—PCN > NCN: Direction signal to FDD set to "0" (step out), and issues step pulses.

The rate at which step pulses are issued is controlled by SRT (Stepping Rate Time) in the SPECIFY command. After each step pulse is issued, NCN is compared against PCN, and when NCN = PCN, then the SE bit in Status Register 0 is set to "1", and the command is terminated.

During the command phase of the seek or recalibrate operation, the 82072 is in the BUSY state, but during the execution phase it is in the NON BUSY state. At this time another SEEK or RECALIBRATE command may be issued, and in this manner, parallel seek operations may be done on up to 4 drives at once.

If an FDD is in a NOT READY state at the beginning of the command execution phase or during the seek operation, then the NR bit in Status Register 0 is set to "1", and the command is terminated.

Note that if implied seek is not enabled, the read and write commands should be preceded by:

1) SEEK command; Step to the proper track

2) SENSE INTERRUPT STATUS command; Terminate the Seek command

3) READ ID. Verify head is on proper track

4) Issue READ/WRITE command.

The SEEK command does not have a result phase. Therefore, it is highly recommended that the SENSE INTERRUPT STATUS Command after the SEEK command be issued to effectively terminate it and to provide verification of the head position (PCN). The H bit (Head Address) in ST0 will always return a "0". When exiting POWERDOWN mode, the 82072 clears the PCN value and the status information to zero. Prior to issuing the POWERDOWN command, it is highly recommended that the user service all pending interrupts through the SENSE INTERRUPT STATUS command.

CONTROL COMMANDS WITHOUT INTERRUPTS

These commands have no execution phase and do not generate an interrupt upon completion. Their primary use is to follow one of the commands that do not have an execution phase, although they may be issued whenever the 82072 is in the command phase.

SENSE INTERRUPT STATUS COMMAND

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	0	0	0	1	0	0	0	Command Code
Result	R	—	—	—	—	ST0	—	—	—	
	R	—	—	—	—	PCN	—	—	—	

An interrupt signal on INT pin is generated by the 82072 for one of the following reasons:

1. Upon entering the Result Phase of:
   a. READ DATA Command
   b. READ TRACK Command
   c. READ ID Command
   d. READ DELETED DATA Command
   e. WRITE DATA Command
   f. FORMAT TRACK Command
   g. WRITE DELETED DATA Command
2. RDY pin changes state
3. End of SEEK, RELATIVE SEEK or RECALIBRATE Command
4. 82072 requires a data transfer during the execution phase in the non-DMA Mode

The SENSE INTERRUPT STATUS command resets the interrupt signal and via the IC code and SE bit of Status Register 0, identifies the cause of the interrupt.

Table 10. Interrupt Identification

SE	IC	Interrupt Due To
0	11	RDY pin changes state
1	00	Normal Termination of SEEK or RECALIBRATE command
1	01	Abnormal Termination of SEEK or RECALIBRATE command

The SEEK, RELATIVE SEEK and the RECALIBRATE commands have no result phase. SENSE INTERRUPT STATUS command must be issued immediately after these commands to terminate them and to provide verification of the head position (PCN). The H (Head Address) bit in ST0 will always return a "0". If a SENSE INTERRUPT STATUS is not issued, the drive, will continue to be BUSY and may effect the operation of the next command.

SENSE DRIVE STATUS

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	0	0	0	0	1	0	0	Command Code
	W	MOT	0	0	0	0	HD5	DSI	DS0	
Result	R	———	———	———	———	ST3	———	———	———	

SENSE DRIVE STATUS obtains drive status information. It has no execution phase and goes directly to the result phase from the command phase. STATUS REGISTER 3 contains the drive status information. If Bit 7 of the second command byte is set (MOT), there is no delay before the drive is accessed for status information. If Bit 7 is not set, and the motor is off or a drive different from the current drive is selected. The 82072 waits the motor ON/OFF delay period before accessing the drive. In either case, the motor is always turned on before the status information is accessed.

SPECIFY COMMAND

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	0	0	0	0	0	1	1	Command Code
	W	<———	SRT	———>		<———	HUT	———>		
	W	<———————			HLT	———————		>	ND	
Execution										Values Placed Into Regs.

The SPECIFY command sets the initial values for each of the three internal timers. The HUT (Head Unload Time) defines the time from the end of the execution phase of one of the read/write commands to the head unload state. The SRT (Step Rate Time) defines the time interval between adjacent step pulses. Note that the spacing between the first and second step pulses may be shorter than the remaining step pulses. The HLT (Head Load Time) defines the time between the Head Load signal goes high and the read, write operation starts. The values change with the data rate speed selection and are documented in Table 11. The values are the same for MFM and FM.

Table 11. Drive Control Delays (ms)

	HUT				SRT			
	1M	500K	300K	250K	1M	500K	300K	250K
1	128	256	512	1024	8.0	16	26.7	32
2	8	16	26.7	32	7.5	15	25	30
..	..	..	..	..	..	..	..	..
E	112	224	373	448	1.0	2	3.33	4
F	120	240	400	480	0.5	1	1.67	2

	HLT			
	1M	500K	300K	250K
00	128	256	512	1024
01	1	2	3.3	4
02	2	4	6.7	8
..	..	..	..	.
7F	126	252	420	504
7F	127	254	423	508

The choice of DMA or NON-DMA operations is made by the ND bit. When this bit is "1", the NON-DMA mode is selected, and when ND is "0", the DMA mode is selected. In DMA mode, data transfers are signalled by the DRQ pin. Non-DMA mode uses the RQM bit and the INT pin to signal data transfers.

NEW CONTROL COMMANDS

CONFIGURE COMMAND

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	0	0	1	0	0	1	1	
	W	HSDA	<— — MOFF — —>			<— — — MON — — >				Configuration
	W	0	EIS	EFIFO	POLL	<— — FIFOTHR— —>				Information
	W	<— — — — — — — — — PRETRK —— — — — — — — — —>								

Issued to select the special features of the 82072. A CONFIGURE command need not be issued if the default values of the 82072 meet the system requirements.

CONFIGURE DEFAULT VALUES:

EIS —No Implied Seeks
EFIFO —FIFO Disabled
POLL —Polling Enabled
FIFOTHR —FIFO Threshold Set to 1 Byte
PRETRK —Pre-Compensation Set to Track 0
HSDA —Disabled
MOFF —Set to Motor Off Delay of 5.2 Seconds
MON —Set to 0.0 Second

EIS

Enable implied seek. When set to "1", the 82072 will perform a SEEK operation before executing a read or write command. Defaults to no implied seek.

EFIFO

A "1" puts the FIFO into the 8272A compatible mode where the FIFO is disabled. This means data transfers are asked for on a byte by byte basis. Defaults to "1", FIFO disabled. The threshold defaults to one.

POLL

Disable polling of the drives. Defaults to "0", polling enabled. When enabled, each of the drives is selected for a period of time, and its RDY pin sampled. After a delay, the next drive is selected. This sequence occurs whenever the 82072 is in the idle state. An interrupt is generated if the 82072 detects a change in the drive RDY signal. No polling is performed while the drive head is loaded and the head unload delay has not expired.

FIFOTHR

The FIFO threshold in the execution phase of read or write commands. This is programmable from 1 to 16 bytes. Defaults to one byte.

00 = 1 byte
01 = 2 bytes
02 = 3 bytes
...
0F = 16 bytes

PRETRK

Pre-compensation start track number. Programmable from track 0 to 255. Defaults to track 0.

00 = TRACK 0
01 = TRACK 1
02 = TRACK 2
...
FE = TRACK 254
FF = TRACK 255

HSDA

High Speed Disk Adjust causes the motor on/off delays to be doubled. This is necessary for disks that rotate at high rates (i.e., 600 RPM vs 300 RPM). Defaults to a "0" which is disabled.

MOFF

Programs the number of index pulses to be counted before the MOTOR pin is deactivated (if high). All 82072 commands restart the motor off timer upon their completion. The MOFF bits increment the count value by 4. The HSDA bit doubles that value for disks that spin extremely fast. This chart is simplified to illustrate the count delays. All combinations are legal and no checking is done by the 82072.

The motor off delay is ignored while any drive is BUSY.

Defaults to 110 (5.2 sec. at 300 RPM).

MOFF Bits	HSDA	Revolutions	Delay 300 RPM	Delay 360 RPM	Delay 600 RPM	
000*	0	2	0.4	0.33	0.2	SEC.
000	1	4	—	—	0.4	
001	0	6	1.2	1.0	0.6	
001	1	12	—	—	1.2	
010	0	10	2.0	1.67	1.0	
010	1	20	—	—	2.0	
•	•	•	•	•	•	
•	•	•	•	•	•	
110	0	26	5.2	4.33	2.6	
110	1	52	—	—	5.2	
111	0	30	6.0	5.0	3.0	
111	1	60	—	—	6.0	

NOTE:
If MOFF and MON and HSDA are all zero, Motor Off Delay is infinite. MOTOR Pin must be given a command to go On/Off.

MON

Programs the number of Index pulses that are counted after the MOTOR pin goes active before a command can proceed into the execution phase. This is to give the floppy drive time to spin up and stabilize at the proper speed. If MOTOR is already active, no delay will be generated even if the same drive is selected. The MON bits increment the index count by one and the HSDA bit will double that value if active. All combinations are legal and no checking is done by the 82072.

Defaults to 0000 (0.0 sec. at 300 RPM).

MON	HSDA	Motor on to Command Execution Delay: Revolutions	300 RPM	360 RPM	600 RPM	
0000		Zero Motor On Delay				
0001	0	1	0.2	0.17	0.1	SECS.
0001	1	2	—	—	0.2	
0010	0	2	0.4	0.33	0.2	
0010	1	4	—	—	0.4	
0011	0	3	0.6	0.5	0.3	
0011	1	6	—	—	0.6	
:	:	:	:	:	:	
1111	0	15	3.0	2.5	1.5	
1111	1	30	—	—	3.0	

NOTE:
Actual index counts depend upon the position of the disk and when the command begins processing. There is an error rate of $-1/+0$ revolutions.

The MOTOR logic provides a programmable motor enable signal to the FDDs. This is to allow the drive time to spin up the floppy and stabilize before attempting a read or write. Most floppy drives do not require the drive to be selected (through the DSx line) or to be ready. The application should qualify the MOTOR pin with the drive select outputs. The 82072 will activate the appropriate drive select outputs along with the MOTOR pin. Figure 14 is an example of the interface logic. If a drive is not currently selected or the MOTOR signal is not on, DS and MOTOR are activated the programmed time before the drive is accessed.

NOTE:
This interface logic assumes polling is disabled. undesired drive selects will occur if polling is enabled.

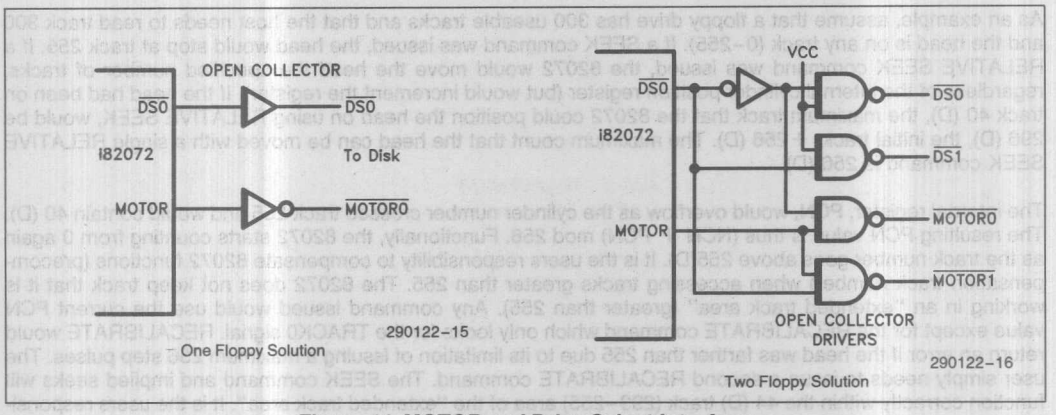


Figure 14. MOTOR and Drive Select Interface

MOTOR ON/OFF COMMAND

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	ON/OFF	DS1	DS0	0	1	0	1	1	Command Code

ON/OFF

Determines whether the command sets the MOTOR pin high or low independent of the internal timer. A "0" takes the MOTOR pin low immediately and a "1" takes the pin to a high. The 82072 returns to the command phase.

The internal timer takes drive changes into account. When switching between drives, the MOFF time will be ignored so that the newly selected drive can be accessed as soon as the MOTOR ON delay has expired. The MOTOR ON/OFF command allows the user to override the programmed delays. The typical implementation would not use the MOTOR ON/OFF command to enable the drive motor because the delays would be the responsibility of the host system.

RELATIVE SEEK

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	1	DIR	0	0	1	1	1	1	
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	<———— RCN ————>								

NOTE:
The command is coded the same as for SEEK, except for the MSB of the first byte and the DIR bit.

DIR Head Step Direction Control.

DIR	Action
0	Step Head Out
1	Step Head In

RCN Relative Cylinder Number that determines how many tracks to step the head in or out from the current track number.

The RELATIVE SEEK command differs from the SEEK command in that it steps the head the absolute number of tracks specified in the command instead of making a comparison against an internal register. The SEEK command is good for drives that support a maximum of 256 tracks. RELATIVE SEEKs cannot be overlapped with other RELATIVE SEEKs. Only one RELATIVE SEEK can be active at a time. RELATIVE SEEKs may be overlapped with SEEKs and RECALIBRATEs. Bit 4 of Status Register 0 (EC) will be set if RELATIVE SEEK attempts to step outward beyond Track 0.

As an example, assume that a floppy drive has 300 useable tracks and that the host needs to read track 300 and the head is on any track (0–255). If a SEEK command was issued, the head would stop at track 255. If a RELATIVE SEEK command was issued, the 82072 would move the head the specified number of tracks, regardless of the internal cylinder position register (but would increment the register). If the head had been on track 40 (D), the maximum track that the 82072 could position the head on using RELATIVE SEEK, would be 296 (D), the initial track, +256 (D). The maximum count that the head can be moved with a single RELATIVE SEEK command is 256 (D).

The internal register, PCN, would overflow as the cylinder number crossed track 255 and would contain 40 (D). The resulting PCN value is thus (NCN + PCN) mod 256. Functionally, the 82072 starts counting from 0 again as the track number goes above 255(D). It is the users responsibility to compensate 82072 functions (precompensation track number) when accessing tracks greater than 255. The 82072 does not keep track that it is working in an "extended track area" (greater than 255). Any command issued would use the current PCN value except for the RECALIBRATE command which only looks for the TRACK0 signal. RECALIBRATE would return an error if the head was farther than 255 due to its limitation of issuing a maximum 256 step pulses. The user simply needs to issue a second RECALIBRATE command. The SEEK command and implied seeks will function correctly within the 44 (D) track (299–255) area of the "extended track area". It is the users responsibility not to issue a new track position that would exceed the maximum track that is present in the extended area.

To return to the standard floppy range (0–255) of tracks, a RELATIVE SEEK would be issued to cross the track 255 boundary.

A RELATIVE SEEK can be used instead of the normal SEEK but the host is required to calculate the difference between the current head location and the new (target) head location. This may require the host to issue a READ ID command to ensure that the head is physically on the track that software assumes it to be. Different 82072 commands will return different cylinder results which may be difficult to keep track of with software without the READ ID command.

DUMPREG

Phase	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Remarks
Command	W	0	0	0	0	1	1	1	0	*Note
Execution										Registers Placed into FIFO
	R	PCN-Drive 0								
	R	PCN-Drive 1								
	R	PCN-Drive 2								
	R	PCN-Drive 3								
Result	R	< — — — SRT — — — >				< — — — HUT — — — >				
	R	< — — — — — — — — HLT — — — — — — — — >							ND	
	R	< — — — — — — — — SC/EOT — — — — — — — — >								See Note
	R	HSDA	< — MOFF — >			< — — — MON — — — >				
	R	0	EIS	EFIFO	POLL	< — — FIFOTHR — — >				
	R	< — — — — — — — — PRETRK — — — — — — — — >								

NOTE:
SC is returned if the last command that was issued was the Format command. EOT is returned if the last command issued was a Read or Write.

The DUMPREG command is designed to support system run-time diagnostics and application software development and debug.

STATUS REGISTER ENCODING

The contents of these registers are available only through a command sequence.

Status Register 0

Bit No.	Symbol	Name	Description
7, 6	IC	Interrupt Code	00-Normal termination of command. The specified command was properly executed and completed without error. 01-Abnormal termination of command. Command execution was started, but was not successfully completed. 10-Invalid command. The requested command could not be executed. 11-Abnormal termination. During the command execution, the RDY signal changed state.
	SE	Seek End	The 82072 completed a SEEK or RECALIBRATE command, or a READ or WRITE with implied seek command.
4	EC	Equipment Check	The TRK0 pin failed to become a "1" after: 1. 256 step pulses in the RECALIBRATE command. 2. The RELATIVE SEEK command causes the 82072 to step outward beyond Track 0.
3	NR	Not Ready	RDY pin became a "0" while executing a read, write, seek, or recalibrate command.
2	H	Head Address	The current head address.
1, 0	DS1, 0	Drive Select	The current selected drive.

Status Register 1

Bit No.	Symbol	Name	Description
7	EN	End of Cylinder	The 82072 tried to access a sector beyond the final sector of the track (255D).
6	—	—	Unused. This bit is always "0".
5	DE	Data Error	The 82072 detected a CRC error in either the ID field or the data field of a sector.
4	OR	Overrun/ Underrun	Becomes set if the 82072 does not receive CPU or DMA service within the required time interval, resulting in data overrun or underrun.
3	—	—	Unused. This bit is always "0".
2	ND	No Data	Any one of the following: 1. READ DATA, READ DELETED DATA command, the 82072 did not find the specified sector. 2. READ ID command, the 82072 cannot read the ID field without an error. 3. READ TRACK command, the 82072 cannot find the proper sector sequence.
1	NW	Not Writable	WP pin became a "1" while the 82072 is executing a WRITE DATA, WRITE DELETED DATA, or FORMAT TRACK command.
0	MA	Missing Address Mark	Any one of the following: 1. The 82072 did not detect an ID address mark at the specified track after encountering the index pulse from the IDX pin twice. 2. The 82072 cannot detect a data address mark or a deleted data address mark on the specified track.

Status Register 2

Bit No.	Symbol	Name	Description
7	—	—	Unused. This bit is always "0".
6	CM	Control Mark	Any one of the following: 1. READ DATA command, the 82072 encounters a deleted data address mark. 2. READ DELETED DATA command, the 82072 encounters a data address mark.
5	DD	Data Error in Data Field.	The 82072 detected a CRC error in the data field.
4	WC	Wrong Cylinder	The track address from the sector ID field is different from the track address maintained inside the 82072. With implied seek, this bit will not be set.
3	—	—	Unused. This bit is always "0".
2	—	—	Unused. This bit is always "0".
1	BC	Bad Cylinder	The track address from the sector ID field is different from the track address maintained inside the 82072 and is equal to FF hex which indicates a bad track with a hard error according to the IBM soft-sectored format.
0	MD	Missing Data Address Mark	The 82072 cannot detect a data address mark or a deleted data address mark.

Status Register 3

Bit No.	Symbol	Name	Description
7	—	—	Unused. This bit is always "0".
6	WP	Write Protected	Indicates the status of the WP pin.
5	RDY	Ready	Indicates the status of the RDY pin.
4	T0	TRACK 0	Indicates the status of the TRK0 pin.
3	—	—	Unused. This bit is always "1".
2	HD	Head Address	Indicates the status of the HDSEL pin.
1, 0	DS1, 0	Direct Select	Indicates the status of the DS1, DS0 pins.

D.C. SPECIFICATION

ABSOLUTE MAXIMUM RATINGS

Storage Temperature ..... −65°C to +150°C
Supply Voltage ..... −0.5 to +8.0V
Voltage on Any Input ..... GND − 2V to 6.5V
Voltage on Any Output .. GND − 0.5V to VCC + 0.5V
Power Dissipation ..... 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

T_A = 0°C to = 70°C, V_{CC} = +5V ±10%, V_{SS} = AV_{SS} = 0V

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Input Low Voltage, X1	−0.5	0.8	V	
V_{IHC}	Input High Voltage, X1	3.9	V_{CC} +0.5	V	
V_{IL}	Input Low Voltage (all pins except X1)	−0.5	0.8	V	
V_{IH}	Input High Voltage (all pins except X1)	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC} −0.4		V V	I_{OH} = −2.5 mA I_{OH} = 100 µA
I_{CC}	V_{CC} Supply Current (Digital and Analog)		tbd tbd 25 35	mA mA mA mA	(Note 1, 3, 5, 6) (Note 1, 4, 5, 6) (Note 2, 3, 5, 6) (Note 2, 4, 5, 6)
I_{CCSB}	I_{CC} in Powerdown		1	mA	(Note 3, 5)
I_{IL}	Input Load Current (all input pins)		10 −10	µA µA	V_{IN} = V_{CC} V_{IN} = 0V
I_{OFL}	Data Bus Output Float Leakage		±10	µA	0.45 < V_{OUT} < V_{CC}

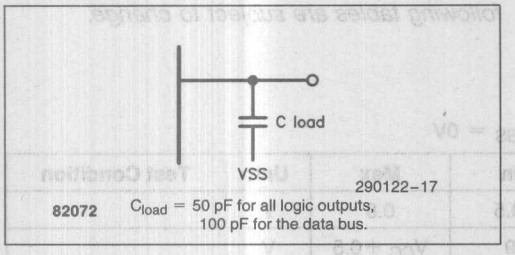
NOTES:
1. Tested at the 1 Mbps data rate.
2. Tested at the 500 kbps data rate.
3. V_{IL} = V_{SS}, V_{IH} = V_{CC}; Outputs not connected to D.C. loads.
4. V_{IL} = 0.8, V_{IH} = 2.0; Outputs not connected to D.C. loads.
5. Each data bus input that is floated can increase the I_{CC} by about 1.5 mA if the 82072 is not in the POWER DOWN mode. These are the only inputs that may be floated.
6. Tested while the 82072 internal PLL is reading a sync field of "00".

Capacitance

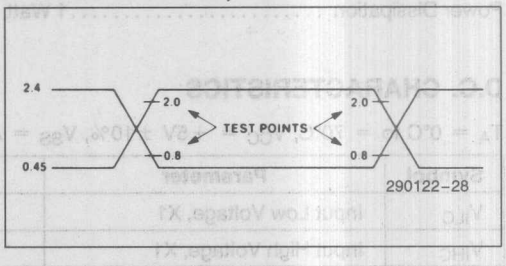
C_{IN}	Input Capacitance	10	pF	F = 1 MHz, T_A = 25°C
C_{IN1}	Clock Input Capacitance	20	pF	Sampled, not 100% Tested
$C_{I/O}$	Input/Output Capacitance	20	pF	

NOTE:
All pins except pins under test are tied to AC ground.

LOAD CIRCUIT



A. C. TESTING INPUT, OUTPUT WAVEFORM



A.C. SPECIFICATIONS

T_A = 0°C to 70°C, V_{CC} = +5V ±10%, V_{SS} = AV_{SS} = 0V

Symbol	Parameter	Min	Max	Unit	Notes
CLOCK TIMINGS					
t1	Clock Rise Time		10	ns	
	Clock Fall Time		10	ns	
t2	Clock High Time	16	26	ns	(Note 7)
t3	Clock Low Time	16	26	ns	(Note 7)
t4	Clock Period	40	43	ns	
t5	Internal Clock Period				(Note 3)
HOST READ CYCLES					
t6	Deleted Specification				
t7	Address Setup to $\overline{RD}$	5		ns	
t8	$\overline{RD}$ Pulse Width	100		ns	
t9	Address Hold From RD	0		ns	
t10	Data Valid From $\overline{RD}$		60	ns	
t11	Command Inactive	60		ns	
t12	Output Float Delay		35	ns	
t13	INT Delay From RD		t5 + 125	ns	
t14	Data Hold From RD	5		ns	

A.C. SPECIFICATIONS (Continued)

T_A = 0°C to 70°C, V_{CC} = +5V ±10%, V_{SS} = AV_{SS} = 0V

Symbol	Parameter	Min	Max	Unit	Notes
HOST WRITE CYCLES					
t15	Address Setup to $\overline{WR}$	5		ns	
t16	$\overline{WR}$ Pulse Width	100		ns	
t17	Address Hold from WR	0		ns	
t18	Command Inactive	60		ns	
t19	Data Setup to WR	70		ns	
t20	Data Hold from WR	0		ns	
t21	INT Delay from WR		t5+125	ns	
DMA CYCLES					
t22	DRQ Cycle Period	6.5		µs	(Note 1)
t23	DACK to DRQ Inactive		75	ns	(Note 1)
t24	RD to DRQ Inactive		150	ns	(Note 4)
t25	DACK Setup to $\overline{RD}$, $\overline{WR}$	5		ns	
t26	DACK Hold from RD, WR	0		ns	
t27	DRQ to $\overline{RD}$, $\overline{WR}$ Active	0	6	µs	(Note 1)
t28	Terminal Count Width	1		t5	
t29	TC to DRQ Inactive		2.5 t5+100	ns	
RESET					
t30	Reset Width	170		t4	(Note 5)
t31	Reset to Control Inactive		2	µs	
WRITE DATA TIMING					
t32	Write Data Width	t5 − 50		ns	(Note 6)
DRIVE CONTROL					
t33	DS0, 1 Setup to DIR	6	25	µs	(Note 8)
t34	DS0, 1 Hold from DIR	8		µs	
t35	DIR Setup to STEP	0.5	2	µs	
t36	DIR Hold from STEP	10		µs	
t37	STEP Active Time (High)	2.5		µs	
t38	STEP Cycle Time			µs	(Note 2)
t39	INDEX Pulse Width	5		t5	

A.C. SPECIFICATIONS (Continued)

T_A = 0°C to 70°C, V_{CC} = +5V ±10%, V_{SS} = AV_{SS} = 0V

Symbol	Parameter	Min	Max	Unit	Notes
READ DATA TIMING					
t40	Read Data Pulse Width	40		ns	
t41	Window Setup to RDDATA	15		ns	
t42	Window Hold from RDDATA	15		ns	
t43	Window Cycle Time	2		µs	FM
	Window Cycle Time	0.5		µs	MFM
f44	PLL Data Rate		1M	bit/s	82072-1
			500K	bit/s	82072
t44	Data Rate Period				1/f44
tLOCK	Lockup Time		64	t44	

NOTES:

1. This timing is for FIFO threshold = 1. When FIFO threshold is N bytes, the value should be multiplied by N and subtract 1.5 µs. The value shown is for 1 Mbps, scales linearly with data rate.
2. This value can be ranged from 0.5 ms to 8.0 ms and is dependent upon data rate.
3. Many timings are a function of the selected data rate. The nominal values for the internal clock period (t5) for the various data rates are:

1 Mbps	3 x oscillator period = 125 ns
500 Kbps	6 x oscillator period = 250 ns
300 Kbps	10 x oscillator period = 420 ns
250 Kbps	12 x oscillator period = 500 ns

4. If $\overline{DACK}$ transitions before $\overline{RD}$, then this specification is ignored. If there is no transition on $\overline{DACK}$, then this becomes the DRQ inactive delay.
5. Reset requires a stable oscillator to meet the minimum active period.
6. Based on the internal clock period (t_5). For various data rates, the Write Data Width minimum values are:

1 Mbps	3 x oscillator period −50 ns = 75 ns
500 Kbps	6 x oscillator period −50 ns = 200 ns
300 Kbps	10 x oscillator period −50 ns = 370 ns
250 Kbps	12 x oscillator period −50 ns = 450 ns

7. Test points for clock high time are 3.5V. Due to transitional times, clock high time max and clock low time max cannot be met simultaneously. Clock high time min and clock low time max cannot be met simultaneously.
8. Based on internal clock period (t_5).

DATA SEPARATOR CHARACTERISTICS

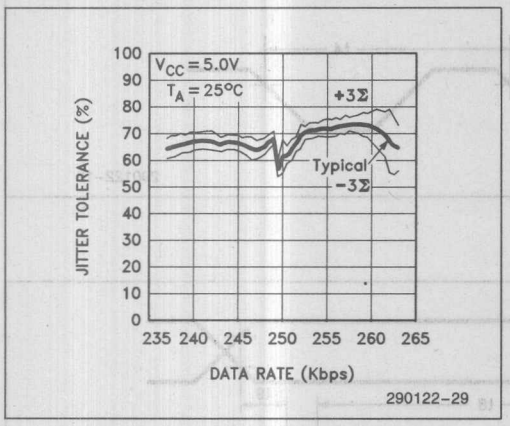


Figure 15. Jitter Tolerance vs Data Rate (F_C = 250 Kbps)

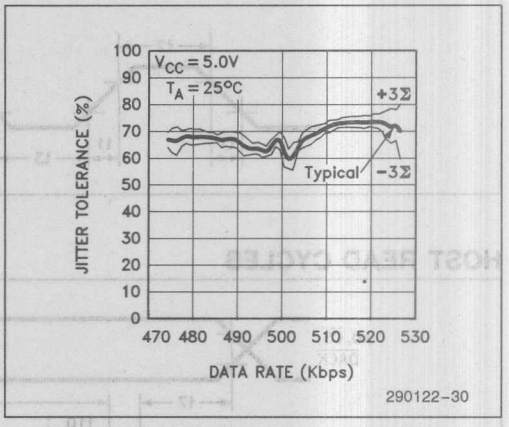


Figure 16. Jitter Tolerance vs Data Rate (F_C = 500 Kbps)

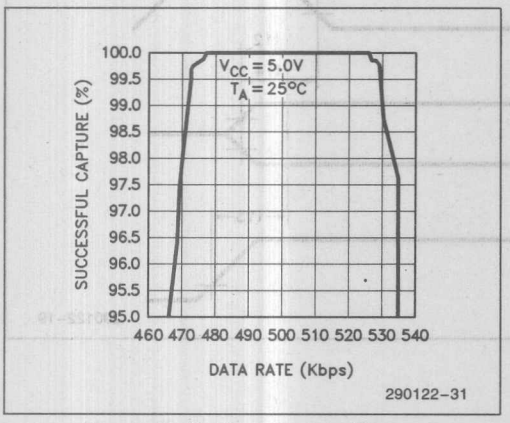


Figure 17. Typical Capture Range (F_C = 250 Kbps)

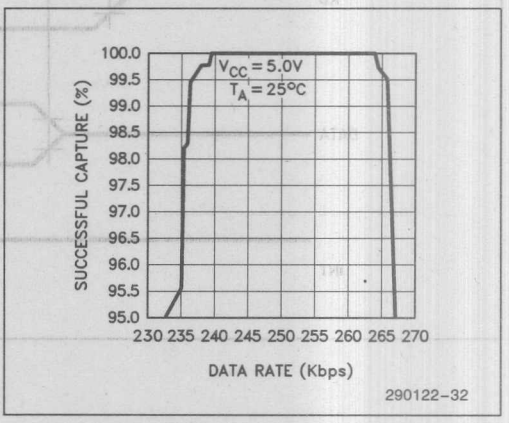


Figure 18. Typical Capture Range (F_C = 500 Kbps)

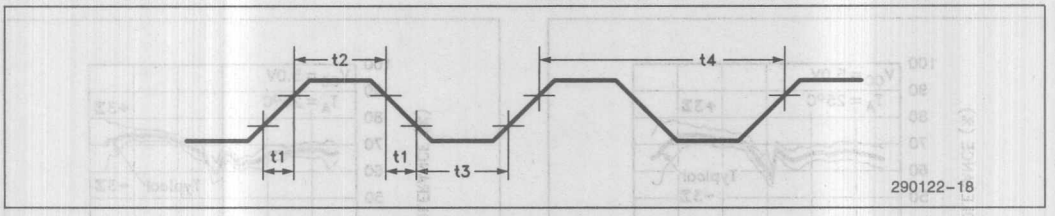
NOTES:

9. Jitter tolerance is defined as: $\frac{\text{Maximum bit shift from nominal position}}{\text{1/4 period of nominal data rate}} \times 100\%$

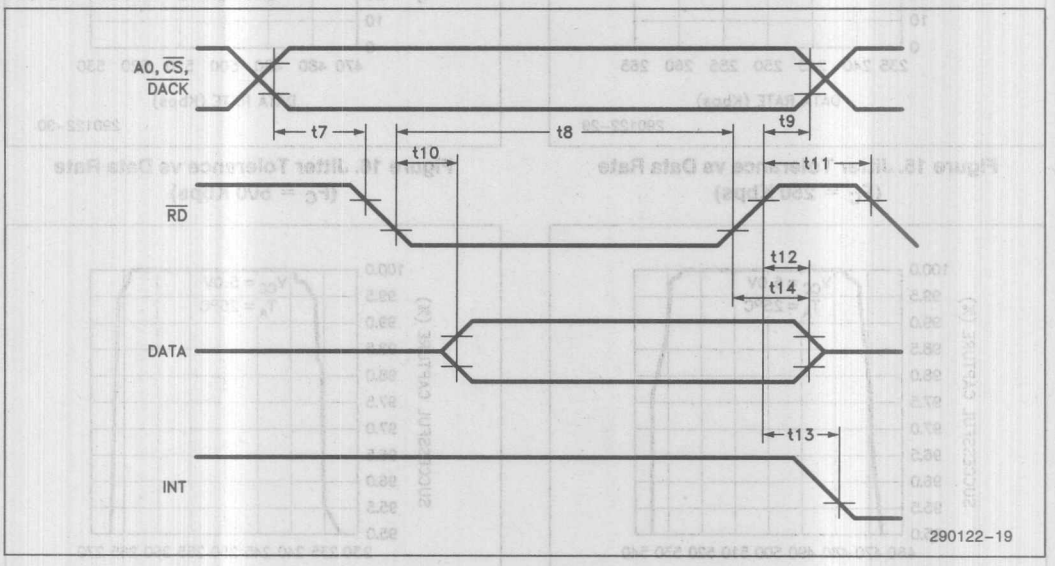
It is a measure of the allowable bit jitter that may be present and still be correctly detected. The data separator jitter tolerance is measured under dynamic conditions that jitters the bit stream according to a reverse precompensation algorithm.

10. Capture range is the frequency range over which the data separator will acquire phase lock with the incoming data stream. This measurement is performed by an iterative process that tabulates the data separator success rate in acquiring lock over a frequency range. The bit stream is jittered by 40% as defined above to reflect typical floppy disk conditions.

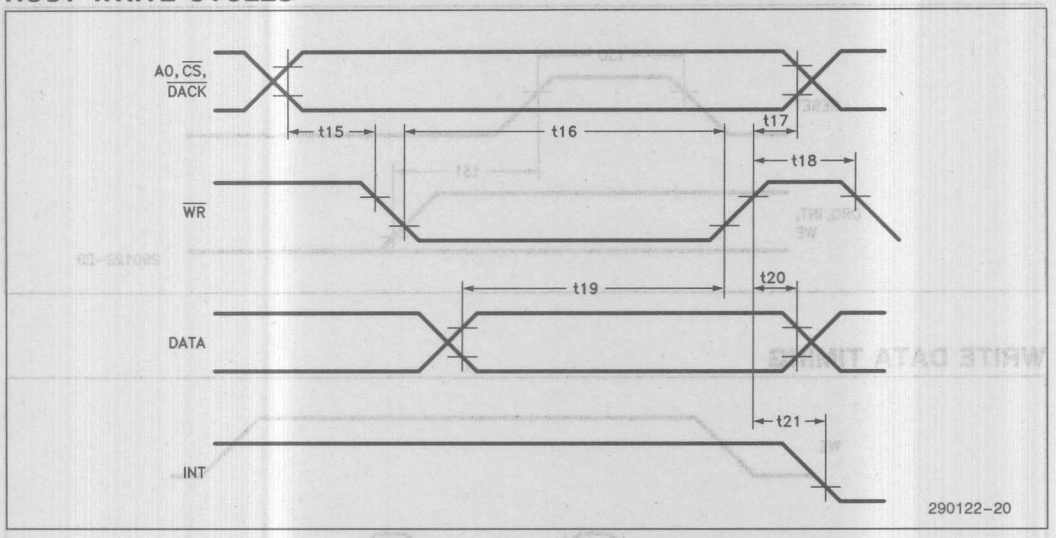
CLOCK TIMING



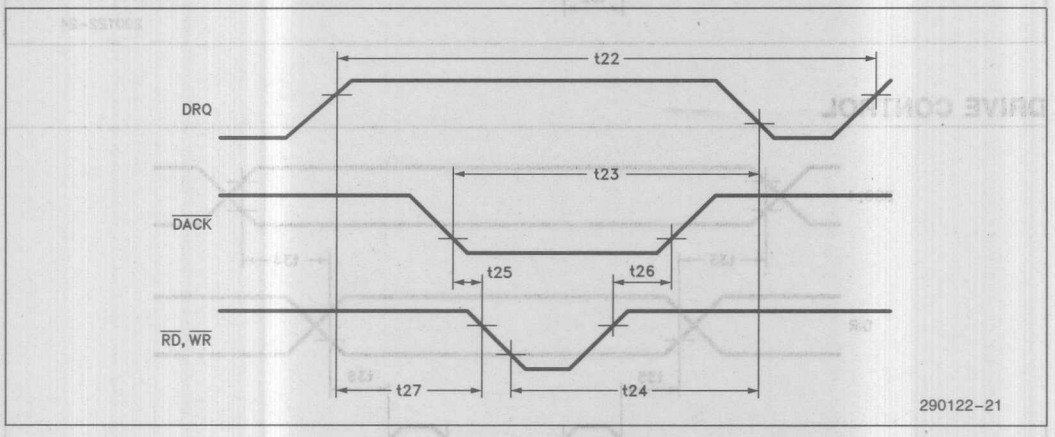
HOST READ CYCLES



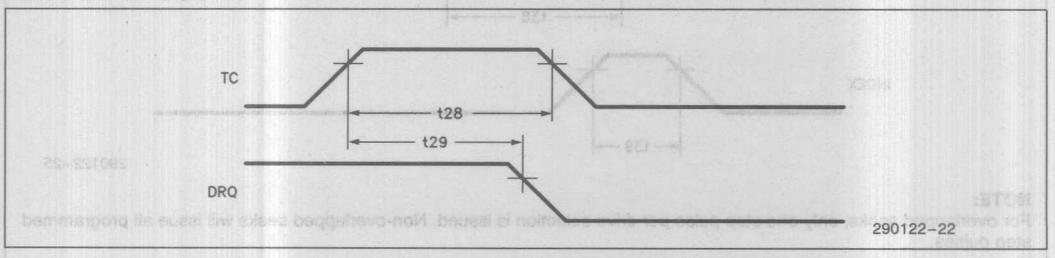
HOST WRITE CYCLES



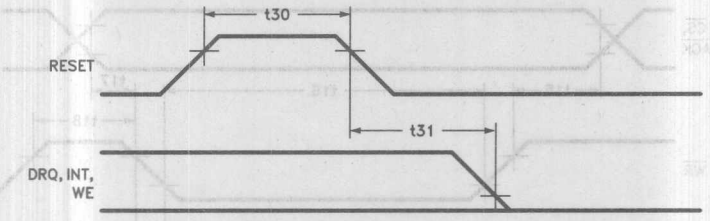
DMA CYCLES



TERMINAL COUNT

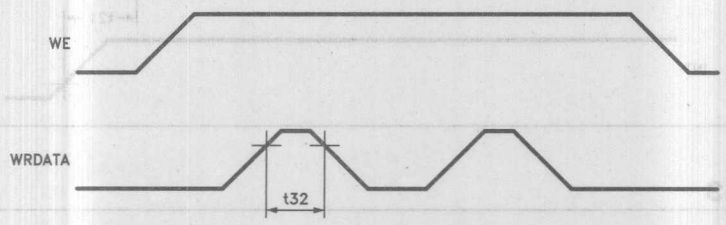


RESET



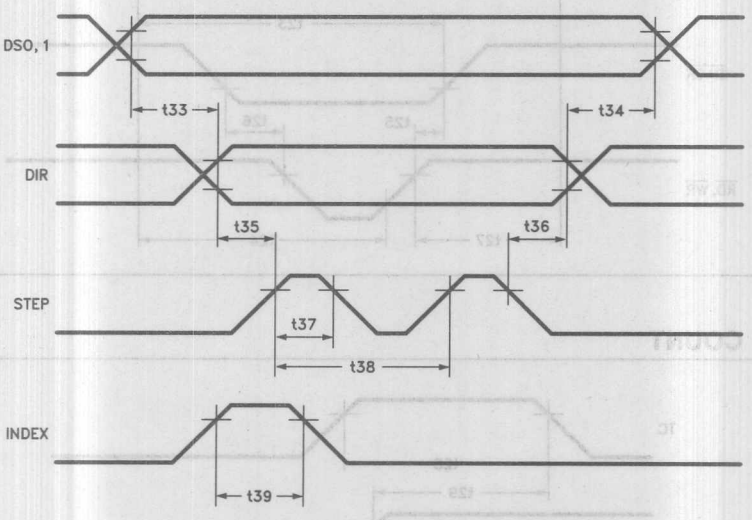
290122-23

WRITE DATA TIMING



290122-24

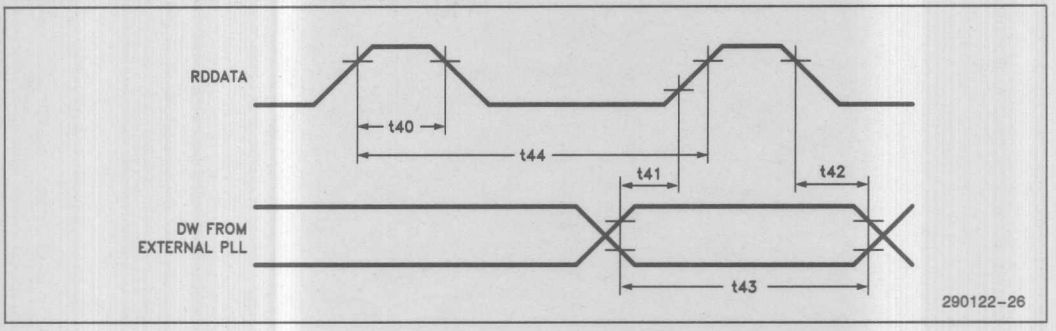
DRIVE CONTROL



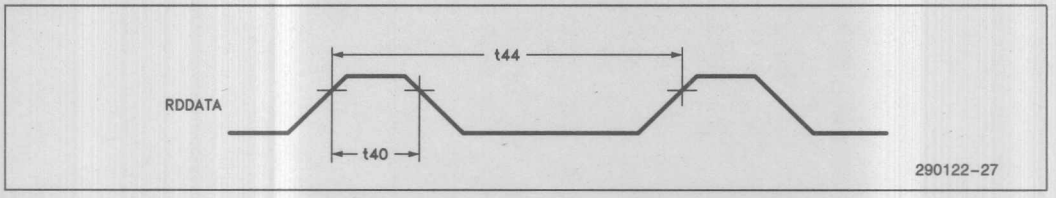
290122-25

NOTE:
For overlapped seeks, only one step pulse per drive selection is issued. Non-overlapped seeks will issue all programmed step pulses.

READ DATA TIMING-EXTERNAL PLL



INTERNAL PLL



READ DATA TIMING-EXTERNAL PLL

INTERNAL PLL

intel

APPLICATION NOTE

AP-116

November 1986

An Intelligent Data Base System Using the 8272

TOM ROSSI
PERIPHERALS APPLICATIONS MANAGER

Order Number: 207875-001

intel

APPLICATION NOTE

AP-116

November 1986

An Intelligent Data Base System Using the 8272

TOM ROSSI
PERIPHERALS APPLICATIONS MANAGER

Order Number: 201816-001

1.0 INTRODUCTION

Most microcomputer systems in use today require low-cost, high-density removable magnetic media for information storage. In the area of removable media, a designer's choice is limited to magnetic types and floppy disks (flexible diskettes), both of which offer non-volatile data storage. The choice between these two technologies is relatively straight-forward for a given application. Since disk drives are designed to permit random access to stored information, they are significantly faster than tape units. For example, locating information on a disk requires less than a second, while tape movement (even at the fastest rewind or fast-forward speed) often requires several minutes. This random access ability permits the use of floppy disks in on-line storage applications (where information must be located, read, and modified/updated in real-time under program or operator control). Tapes, on the other hand, are ideally suited to archival or back-up storage due to their large storage capacities (more than 10 million bytes of data can be archived on a cartridge tape).

A sophisticated controller is required to capitalize on the abilities of the disk storage unit. In the past, disk controller designs have required upwards of 150 ICs. Today, the single-chip 8272 Floppy Disk Controller (FDC) plus approximately 30 support devices can handle up to four million bytes of on-line data storage on four floppy disk drives.

The Floppy Disk

A floppy disk is a circular piece of thin plastic material covered with a magnetic coating and enclosed in a protective jacket (Figure 1). The circular piece of plastic revolves at a fixed speed (approximately 360 rpm) within its jacket in much the same manner that a record revolves at a fixed speed on a stereo turntable. Disks are manufactured in a variety of configurations for various storage capacities. Two standard physical disk sizes are commonly used. The 8-inch disk (8 inches square) is the larger of the two sizes; the smaller size (5¼ inches square) is often referred to as a mini-floppy. Single-sided disks can record information on only one side of the disk, while double-sided disks increase the storage capacity by recording on both sides. In addition, disks are classified as single-density or double-density. Double-density disks use a modified recording method to store twice as much information in the same disk area as can be stored on a single-density disk. Table 1 lists storage capacities for standard floppy disk media.

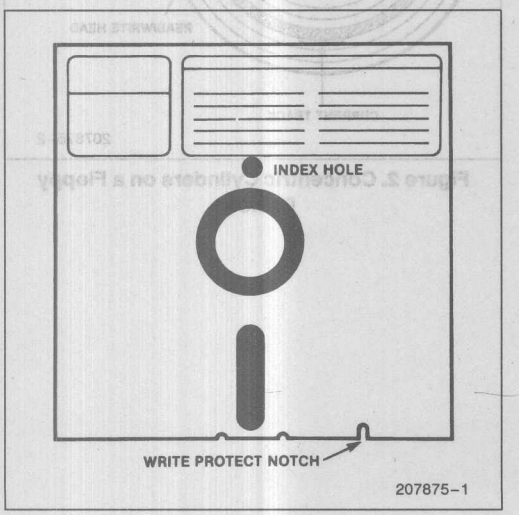


Figure 1. A Floppy Diskette

A magnetic head assembly (in contact with the disk) writes information onto the disk surface and subsequently reads the data back. This head assembly can move from the outside edge of the disk toward the center in fixed increments. Once the head assembly is positioned at one of these fixed positions, the head can read or write information in a circular path as the disk revolves beneath the head assembly. This method divides the surface into a fixed number of cylinders (as shown in Figure 2). There are normally 77 cylinders on a standard disk. Once the head assembly is positioned at a given cylinder, data may be read or written on either side of the disk. The appropriate side of the disk is selected by the read/write head address (zero or one). Of course, a single-sided disk can only use head zero. The combination of cylinder address and head address uniquely specifies a single circular track on the disk. The physical beginning of a track is located by means of a small hole (physical index mark) punched through the plastic near the center of the disk. This hole is optically sensed by the drive on every revolution of the disk.

Table 1. Formatted Disk Capacities

Single-Density Format				
Byte/Sector	128	256	512	1024
Sectors/Track	26	15	8	4
Track/Disk	77	77	77	77
Bytes/Disk	256,256	295,680	315,392	315,392
Double-Density Format				
Byte/Sector	128	256	512	1024
Sectors/Track	52	30	16	8
Track/Disk	77	77	77	77
Bytes/Disk	512,512	591,360	630,784	630,784

Each track is subdivided into a number of sectors (see detailed discussion in section 3). Sectors are generally

128, 256, 512, or 1024 data bytes in length. This track sectoring may be accomplished by one of two techniques: hard sectoring or soft sectoring. Hard sectored disks divide each track into a maximum of 32 sectors. The beginning of each sector is indicated by a sector hole punched in the disk plastic. Soft sectoring, the IBM standard method, allows software selection of sector sizes. With this technique, each data sector is preceded by a unique sector identifier that is read/written by the disk controller.

A floppy disk may also contain a write protect notch punched at the edge of the outer jacket of the disk. This notch is detected by the drive and passed to the controller as a write protect signal.

The Floppy Disk Drive

The floppy disk drive is an electromechanical device that records data on, or reads data from, the surface of a floppy disk. The disk drive contains head control electronics that move the head assembly one increment (step) forward (toward the center of the disk) or backward (toward the edge of the disk). Since the recording head must be in contact with the disk material in order to read or write information, the disk drive also contains head-load electronics. Normally the read/write head is unloaded until it is necessary to read or write information on the floppy disk. Once the head assembly has been positioned over the correct track on the disk, the head is loaded (brought into contact with the disk). This sequence prevents excessive disk wear. A small time penalty is paid when the head is loaded. Approximately thirty to fifty milliseconds are needed before data may be reliably read from, or written to, the disk. This time is known as the head load time. If desired, the head may be moved from cylinder to cylinder while loaded. In this manner, only a small time interval (head settling time) is required before data may be read from the new cylinder. The head settling time is often shorter than the head load time. Typically, disk drives also contain drive select logic that allows more than one physical drive to be connected to the same interface cable (from the controller). By means of a jumper on the drive, the drive number may be selected by the OEM or end user. The drive is enabled only when selected; when not selected, all control signals on the cable are ignored.

Finally, the drive provides additional signals to the system controller regarding the status of the drive and disk. These signals include:

Drive Ready—Signals the system that the drive door is closed and that a floppy disk is inserted into the drive.

Track Zero—Indicates that the head assembly is located over the outermost track of the disk. This signal may be used for calibration of the disk drive at system initialization and after an error condition.

Write Protect—Indicates that the floppy disk loaded into the drive is write protected.

Dual Sided—Indicates that the floppy disk in the drive is dual-sided.

Write Fault—Indicates that an error occurred during a recording operation.

Index—Informs the system that the physical index mark of the floppy disk (signifying the start of a data track) has been sensed.

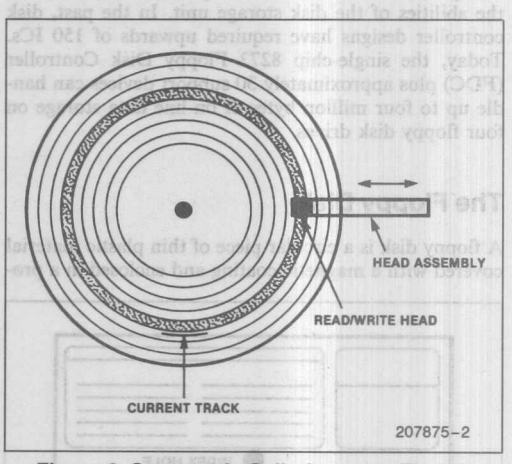


Figure 2. Concentric Cylinders on a Floppy Diskette

2.0 SUBSYSTEM OVERVIEW

A disk subsystem consists of the following functional electronic units:

1) Disk Controller Electronics
2) Disk Drive Electronics
3) Controller/Disk Interface (cables, drivers, terminators)
4) Controller/Microprocessor System Interface

The operation of these functional units is discussed in the following paragraphs.

Controller Electronics

The disk controller is responsible for converting high-level disk commands (normally issued by software executing on the system processor) into disk drive commands. This function includes:

1) Disk Drive Selection—Disk controllers typically manage the operations of multiple floppy disk drives. This controller function permits the system processor to specify which drive is to be used in a particular operation.
2) Track Selection—The controller issues a timed sequence of step pulses to move the head from its current location to the proper disk cylinder from which data is to be read or to which data is to be written. The controller stores the current cylinder number and computes the stepping distance from the current cylinder to the specified cylinder. The controller also manages the head select signal to select the correct side of the floppy disk.
3) Sector Selection—The controller monitors the data on a track until the requested sector is sensed.
4) Head Loading—The disk controller determines the times at which the head assembly is to be brought in contact with the disk surface in order to read or write data. The controller is also responsible for waiting until the head has settled before reading or writing information. Often the controller maintains the head loaded condition for up to 16 disk revolutions (approximately 2 seconds) after a read or write operation has been completed. This feature eliminates the head load time during periods of heavy disk I/O activity.
5) Data Separation—The actual signal recorded on a floppy disk is a combination of timing information (clock) and data. The serial READ DATA input (from the disk drive) must be converted into two signal streams: clock and data. (The READ DATA input operates at 250K bits/second for single-density disks and 500K bits/second for double-density disks.) The serial data must also be assembled into 8-bit bytes for transfer to system memory. A byte must be assembled and transferred every 32 microseconds for single-density disks and every 16 microseconds for double-density.
6) Error-Checking—Information recorded on a floppy disk is subject to both hard and soft errors. Hard (permanent) errors are caused by media defects. Soft errors, on the other hand, are temporary errors caused by electromagnetic noise or mechanical interference. Disk controllers use a standard error checking technique known as a Cyclic Redundancy Check (CRC). As data is written to a disk, a 16-bit CRC character is computed and also stored on the disk. When the data is subsequently read, the CRC character allows the controller to detect data errors. Typically, when CRC errors are detected, the controlling software retries the failed operation (attempting to recover from a soft error). If data cannot reliably be read or written after a number of retries, the system software normally reports the error to the operator. Multiple CRC errors normally indicate unrecoverable media error on the current disk track. Subsequent recovery attempts must be defined by the system deisgners and tailored to meet system interfacing requirements.

Today, single-chip digital LSI floppy disk controllers such as the 8272 perform all the above functions with the exception of data separation. A data separation circuit (a combination of digital and analog electronics) synchronizes itself to the actual data rate of the disk drive. This data rate varies from drive to drive (due to mechanical factors such as motor tolerances) and varies from disk to disk (due to temperature effects). In order to operate reliably with both single- and double-density storage, the data separation circuit must be based on phase-locked loop (PLL) technology. The phase-locked loop data separation logic is described in section 5. The

separation logic, after synchronizing with the data stream, supplies a data window to the LSI disk controller. This window differentiates data information from clock information within the serial stream. The controller uses this window to reconstruct the data previously recorded on the floppy disk.

Drive Electronics

Each floppy disk drive contains digital electronic circuits that translate TTL-compatible command signals into electromechanical operations (such as drive selection and head movement/loading) and that sense and report disk or drive status to the controller (e.g., drive ready, write fault, and write protect). In addition, the drive electronics contain analog components to sense, amplify, and shape data pulses read from, or written to, the floppy disk surface by the read/write head.

Controller/Drive Interface

The controller/drive interface consists of high-current line drivers, Schmitt triggered input gates, and flat or twisted pair cable(s) to connect the disk drive electronics to the controller electronics. Each interface signal line is resistively terminated at the end of the cable farthest from the line drivers. Eight-inch drives may be directly interfaced by means of 50-conductor flat cable. Generally, cable lengths should be less than ten feet in order to maintain noise immunity.

Normally, provisions are made for up to four disk drives to share the same interface cable. The controller may operate as many cable assemblies as practical. LSI floppy disk controllers typically operate one to four drives on a single cable.

Processor/Memory Interface

The disk controller must interface to the system processor and memory for two distinct purposes. First, the processor must specify disk control and command parameters to the controller. These parameters include the selection of the recording density and specification of disk formatting information (discussed in section 3). In addition to disk parameter specification, the processor must also send commands (e.g., read, write, seek, and scan) to the controller. These commands require the specification of the command code, drive number, cylinder address, sector address, and head address. Most LSI controllers receive commands and parameters by means of processor I/O instructions.

In addition to this I/O interface, the controller must also be designed for high-speed data transfer between memory and the disk drive. Two implementation methods may be used to coordinate this data transfer. The lower-cost method requires direct processor intervention in the transfer. With this method, the controller issues an interrupt to the processor for each data transfer. (An equivalent method allows the processor to poll an interrupt flag in the controller status word.) In the case of a disk write operation, the processor writes a data byte (to be encoded into the serial output stream) to the disk controller following the receipt of each controller interrupt. During a disk read operation, the processor reads a data byte (previously assembled from the input data stream) from the controller after each interrupt. The processor must transfer a data byte from the controller to memory or transfer a data byte from memory to the disk controller within 16 or 32 microseconds after each interrupt (double-density and single-density response times, respectively).

If the system processor must service a variety of other interrupt sources, this interrupt method may not be practical, especially in double-density systems. In this case, the disk controller may be interfaced to a Direct Memory Access (DMA) controller. When the disk controller requires the transfer of a data byte, it simply activates the DMA request line. The DMA controller interfaces to the processor and, in response to the disk controller's request, gains control of the memory interface for a short period of time—long enough to transfer the requested data byte to/from memory. See section 6 for a detailed DMA interface description.

3.0 DISK FORMAT

New floppy disks must be written with a fixed format by the controller before these disks may be used to store data. Formatting is a method of taking raw media and adding the necessary information to permit the controller to read and write data without error. All formatting is performed by the disk controller on a track-by-track basis under the direction of the system processor. Generally, a track may be formatted at any time. However, since formatting "initializes" a complete disk track, all previously written data is lost (after a format operation). A format operation is normally used only when initializing new floppy disks. Since soft-sectoring in such a predominant formatting technique (due to IBM's influence), the following discussion will limit itself to self-sectored formats.

Data Recording Techniques

Two standard data recording techniques are used to combine clock and data information for storage on a floppy disk. The single-density technique is referred to as FM encoding. In FM encoding (see Figure 3), a double frequency encoding technique is used that inserts a data bit between two adjacent clock bits. (The presence of a data bit represents a binary "one" while the absence of a data bit represents a binary "zero.") The two adjacent clock bits are referred to as a bit cell, and

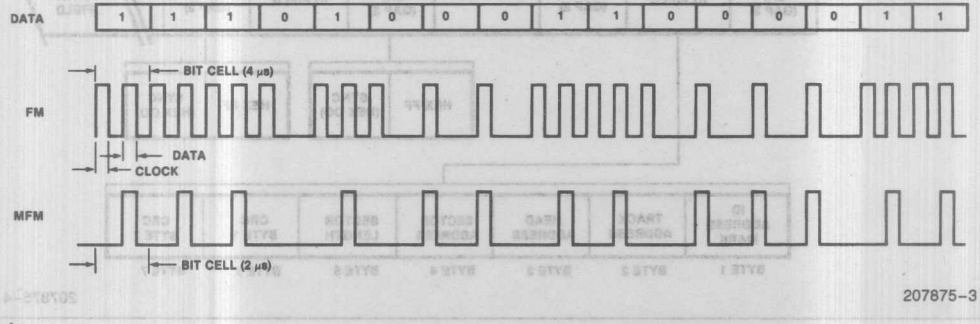
except for a unique field identifiers, all clock bits written on the disk are binary "ones." In FM encoding, each data bit is written at the center of the bit cell and the clock bits are written at the leading edge of the bit cell.

The encoding used for double-density recording is terms MFM encoding (for "Modified FM"). In MFM encoding (Figure 3) the data bits are again written at the center of the bit cell. However, a clock bit is written at the leading edge of the bit cell only if no data bit was written in the previous bit cell and no data bit will be written in the present bit cell.

Sectors

Soft-sectored floppy disks divide each track into a number of data sectors. Typically, sector sizes of 128, 256, 512, or 1024 data bytes are permitted. The sector size is specified when the track initially formatted by the controller. Table 1 lists the single- and double-density data storage capacities for each of the four sector sizes. Each sector within a track is composed of the following four fields (illustrated in Figure 4):

1) Sector ID Field—This field, consisting a seven bytes, is written only when the track is formatted. The ID field provides the sector identification that is used by the controller when a sector must be read or written. The first byte of the field is the ID address mark, a unique coding that specifies the beginning of the ID field. The second, third, and fourth bytes are the cylinder, head, and sector addresses, respectively, and the fifth byte is the sector length code. The last two bytes are the 16-bit CRC character for the ID field. During formatting, the controller supplies the address mark. The cylinder, head, and sector addresses and the sector length code are supplied to the controller by the processor software. The CRC character is derived by the controller from the data in the first five bytes.
2) Post ID Field Gap—The post ID field gap (gap 2) is written initially when the track is formatted. During subsequent write operations, the drive's write circuitry is enabled within the gap and the trailing bytes of the gap are rewritten each time the sector is updated (written). During subsequent read operations, the trailing bytes of the gap are used to synchronize the data separator logic with the upcoming data field.
3) Data Field—The length (number of data bytes) of the data field is determined by software when the track is formatted. The first byte of the data field is the data address mark, a unique coding that specifies the beginning of the data field. When a sector is to be deleted, (e.g., a hard error on the disk), a deleted data address mark is written in place of the data address mark. The last two bytes of the data field comprise the CRC character.
4) Post Data Field Gap—The post data field gap (gap 3) is written when the track is formatted and separates the preceding data field from the next physical ID field on the track. Note that a post data field gap is not written following the last physical sector on a track. The gap itself contains a program-selectable number of bytes. Following a sector update (write) operation, the drive's write logic is disabled during the gap. The actual size of gap 3 is determined by the maximum number of data bits that can be recorded on a track, the number of sectors per track and the total sector size (data plus overhead information). The gap size must be adjusted so that it is large enough to contain the discontinuity generated on the floppy disk when the write current is turned on or off (at the start or completion of a disk write operation) and to contain a synchronization field for the upcoming ID field (of the next sector). On the other hand, the gaps must be small enough so that the total number of data bits required on the track (sectors plus gaps) is less than the maximum number of data bits that can be recorded on the track. The gap



Note:
The FM bit cell is twice the size of the MFM bit cell. Thus, the FM time scale in this figure is 4 μs/bit while the MFM time scale is 2 μs/bit.

Figure 3. FM and MFM Encoding

size must be specified for all read, write, and format operation. The gap size used during disk reads and writes must be smaller than the size used to format the disk to avoid the splice points between contiguous physical sectors. Suggested gap sizes are listed in Table 9.

Tracks

The overall format for a track is illustrated in Figure 4. Each track consists of the following fields:

1) Pre-Index Gap—The pre-index gap (gap 5) is written only when the track is formatted.
2) Index Address Mark—The index address mark consists of a unique code that indicates the beginning of a data track. One index mark is written on each track when the track is formatted.
3) Post Index Gap—The post index gap (gap 1) is used during disk read and write operations to synchronize the data separator logic with the data to be read from the ID field (of the first sector). The post index gap is written only when the disk is formatted.
4) Sectors—The sector information (discussed above) is repeated once for each sector on ths track.

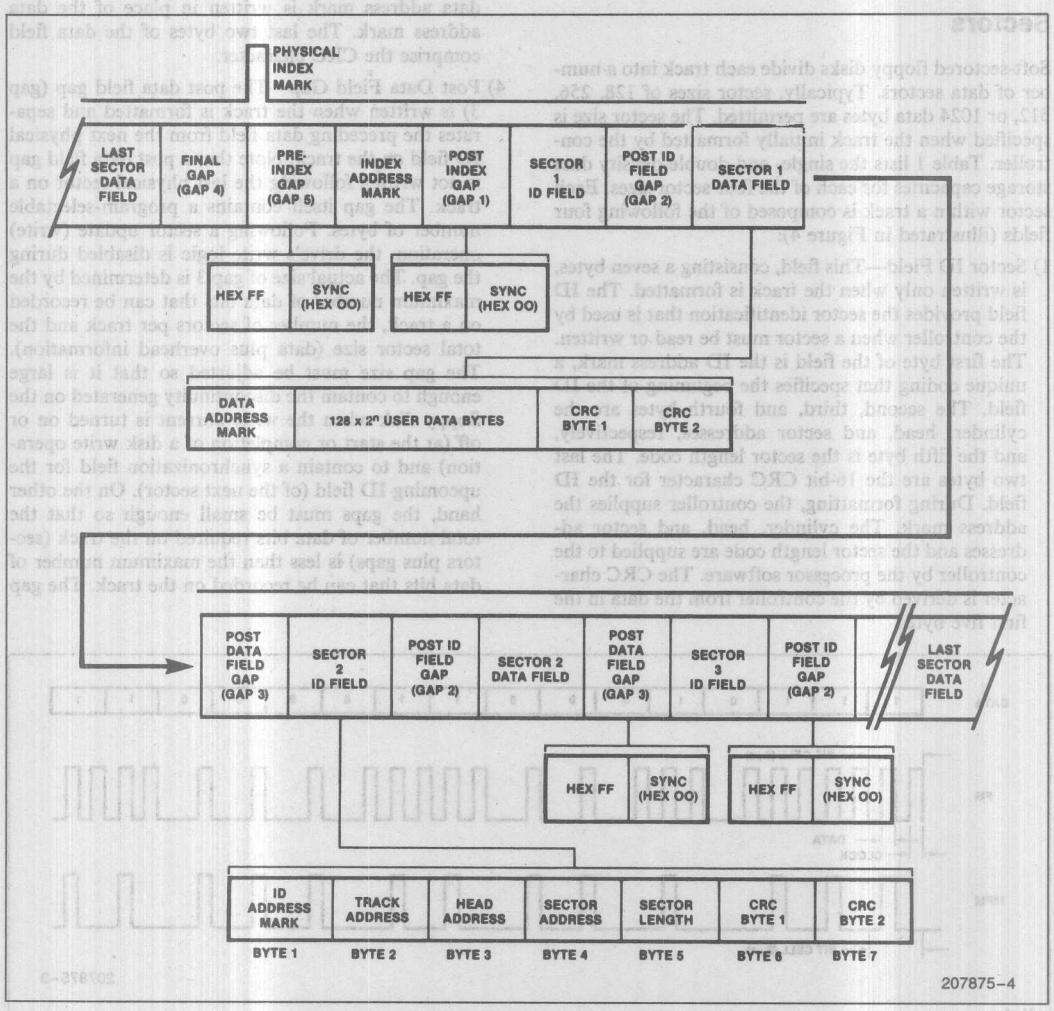


Figure 4. Standard Floppy Diskette Track Format (from SBC 204 Manual)

5) Final Gap—The final gap (gap 4) is written when the track is formatted and extends from the last physical data field on the track to the physical index mark. The length of this gap is dependent on the number of bytes per sector specified, the legnths of the program-selectable gaps specified, and the drive speed.

Sector Interleaving

The initial formatting of a floppy disk determines where sectors are located within a track. It is not necessary to allocate sectors sequentially around the track (i.e., 1,2,3,. . .,26). In fact, it is often advantageous to place the sectors on the track in non-sequential order. Sequential sector ordering optimizes sector access times during multi-sector transfers (e.g., when a program is loaded) by permitting the number of sector specified (up to an entire track) to be transferred within a single revolution of the disk. A technique known as sector interleaving optimizes access times when, although sectors are accessed sequentially, a small amount of processing must be performed between sector reads/writes. For example, an editing program performing a text search reads sectors sequentially, and after each sector is read, performs a software search. If a match is not found, the software issues a read request for the next sector. Since the floppy disk continues to rotate during the time that the software executes, the next physical sector is already passing under the read/write head when the read request is issued, and the processor must wait for another complete revolution of the disk (approximately 166 milliseconds) before the data may actually be input. With interleaving, the sectors are not stored sequentially on a track; rather, each sector is physically removed from the previous sector by some number (known as the interleave factor) of physical sectors as shown in Figure 5. This method of sector allocation provides the processor additional execution time between sectors on the disk. For example, with a 26 sector/track format, an interleave factor of 2 provides 6.4 milliseconds of processing time between sequential 128 byte sector accesses.

To calculate the correct interleave factor, the maximum processor time between sector operations must be divided by the time required for a complete sector to pass under the disk read/write head. After determining the interleave factor, the correct sector numbers are passed to the disk controller (in the exact order that they are to physically appear on the track) during the execution of a format operation.

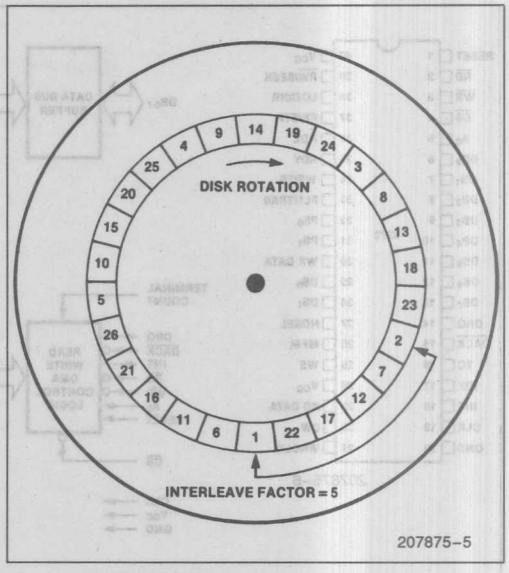


Figure 5. Interleaved Sector Allocation within a Track

4.0 THE 8272 FLEXIBLE DISKETTE CONTROLLER

The 8272 is a single-chip LSI Floppy Disk Controller (FDC) that contains the circuitry necessary to implement both single- and double-density floppy disk storage subsystems (with up to four dual-sided disk drives per FDC). The 8272 supports the IBM 3740 single-density recording format (FM) and the IBM System 34 double-density recording format (MFM). With the 8272, less than 30 ICs are needed to implement a complete disk subsystem. The 8272 accepts and executes high-level disk commands such as format track, seek, read sector, write sector, and read track. All data synchronization and error checking is automatically performed by the FDC to ensure reliable data storage and subsequent retrieval. External logic is required only for the generation of the FDC master clock and write clock (see Section 6) and for data separation (Section 5). The FDC provides signals that control the startup and base frequency selection of the data separator. These signals greatly ease the design of a phase-locked loop data separator.

In addition to the data separator interface signals, the 8272 also provides the necessary signals to interface to

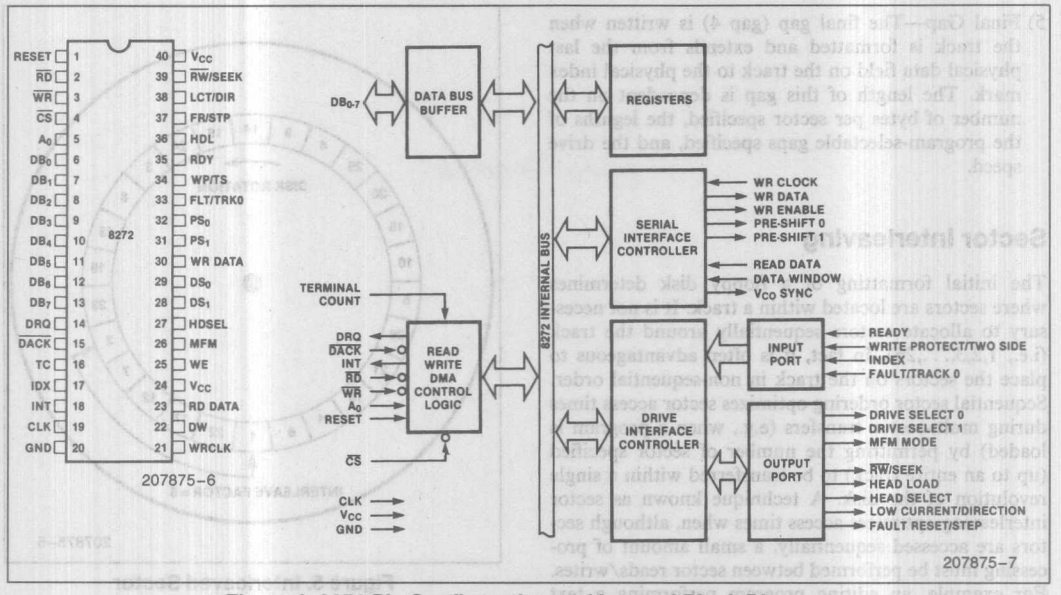


Figure 6. 8272 Pin Configuration and Internal Block Diagram

microprocessor systems with or without Direct Memory Access (DMA) capabilities. In order to interface to a large number of commercially available floppy disk drives, the FDC permits software specification of the track stepping rate, the head load time, and the head unload time.

The pin configuration and internal block diagram of the 8272 is shown in Figure 6. Table 2 contains a description for each FDC interface pin.

Floppy Disk Commands

The 8272 executes fifteen high-level disk interface commands:

Specify	Write Data
Sense Drive Status	Write Deleted Data
Sense Interrupt Status	Read Track
Seek	Read ID
Recalibrate	Scan Equal
Format Track	Scan High or Equal
Read Data	Scan Low or Equal
Read Deleted Data	

Each command is initiated by a multi-byte transfer from the processor to the FDC (the transferred bytes contain command and parameter information). After complete command specification, the FDC automatically executes the command. The command result data (after execution of the command) may require a multi-byte transfer of status information back to the processor. It is convenient to consider each FDC command as consisting of the following three phases:

COMMAND PHASE: The executing program transfers to the FDC all the information required to perform a particular disk operation. The 8272 automatically enters the command phase after RESET and following the completion of the result phase (if any) of a previous command.

EXECUTION PHASE: The FDC performs the operation as instructed. The execution phase is entered immediately after the last command parameter is written to the FDC in the preceding command phase. The execution phase normally ends when the last data byte is transferred to/from the disk (signalled by the TC input to the FDC) or when an error occurs.

RESULT PHASE After completion of the disk operation, status and other housekeeping information are made available to the processor. After the processor reads this information, the FDC re-enters the command phase and is ready to accept another command.

Table 2. 8272 FDC Pin Description

Number	Pin Symbol	I/O	To/From	Description
1	RST	I	μP	**RESET:** Active-high signal that places the FDC in the "idle" state and all disk drive output signals are forced inactive (low). This input must be held active during power on reset while the RD and WR input are active.
2	$\overline{RD}$	I*	μP	**READ:** Active-low control signal that enables data transfer from the FDC to the data bus.
3	$\overline{WR}$	I*	μP	**WRITE:** Active-low control signal that enables data transfer from the data bus into the FDC.
4	$\overline{CS}$	I	μP	**CHIP SELECT:** Active-low control signal that selects the FDC. No reading of writing will occur unless the FDC is selected.
5	A_0	I*	μP	**ADDRESS:** Selects the Data Register or Main Status Register for input/output in conjunction with the RD and WR inputs. (See Table 3.)
6–13	DB_0–DB_7	I/O*	μP	**DATA BUS:** Bidirectional three-state 8-bit data bus.
14	DRQ	O	DMA	**DMA REQUEST:** Active-high output that indicates an FDC request for DMA services.
15	$\overline{DACK}$	I	DMA	**DMA ACKNOWLEDGE:** Active-low control signal indicating that the requested DMA transfer is in progress.
16	TC	I	DMA	**TERMINAL COUNT:** Active-high signal that causes the termination of a command. Normally, the terminal count input is directly connected to the TC/EOP output from the DMA controller, signalling that the DMA transfer has been completed. In a non-DMA environment, the processor must count data transfers and supply a TC signal to the FDC.
17	IDX	I	Drive	**INDEX:** Indicates detection of the physical index mark (the beginning of a track) on the selected disk drive.
18	INT	O	μP	**INTERRUPT REQUEST:** Active-high signal indicating an 8272 interrupt service request.
19	CLK	I		**CLOCK:** Signal phase 8 MHz clock (50% duty cycle).
20	GND			**GROUND:** DC power return.
21	WR CLK	I		**WRITE CLOCK:** 500 kHz (FM) or 1 MHz (MFM) write clock with a constant pulse width of 250 ns (for both FM and MFM recording). The write clock must be present at all times.
22	DW	I	PLL	**DATA WINDOW:** Data sample signal from the phase-locked loop indicating that the FDC should sample input data from the disk drive.
23	RD DATA	I	Drive	**READ DATA:** FDC input data from the selected disk drive.
24	VCO	O	PLL	**VCO SYNC:** Active-high output that enables the phase-locked loop to synchronize with the input data from the disk drive.
25	WE	O	Drive	**WRITE ENABLE:** Active-high output that enables the disk drive write gate.
26	MFM	O	PLL	**MFM MODE:** Active-high output used by external logic to enable the MFM double-density recording mode. When the MFM output is low, single-density FM recording is indicated.
27	HDSEL	O	Drive	**HEAD SELECT:** Selects head 0 or head 1 on a dual-sided disk.

Table 2. 8272 FDC Pin Description (Continued)

Number	Pin Symbol	I/O	To/From	Description
28, 29	DS_1, DS_0	O	Drive	**DRIVE SELECT:** Selects one of four disk drives.
30	WR DATA	O	Drive	**WRITE DATA:** Serial data stream (combination of clock and data bits) to be written on the disk.
31, 32	PS_1, PS_0	O	Drive	**PRECOMPENSATION (PRE-SHIFT) CONTROL:** Write precompensation output control during MFM mode. Specifies early, late, and normal timing signals. See the discussion in Section 5.
33	FLT/TRKO	I	Drive	**FAULT/TRACK 0:** Senses the disk drive fault condition in the Read/Write mode and the Track 0 condition in the Seek mode.
34	WP/TS	I	Drive	**WRITE PROTECT/TWO-SIDED:** Sense the disk write protect status in the Read/Write mode and the dual-sided media status in the Seek mode.
35	RDY	I	Drive	**READY:** Senses the disk drive ready status.
36	HDL	O	Drive	**HEAD LOAD:** Loads the disk drive read/write head. (The head is placed in contact with the disk.)
37	FR/STP	O	Drive	**FAULT RESET/STEP:** Resets the fault flip-flop in the disk drive when operating in the Read/Write mode. Provides head step pulses (to move the head from one cylinder to another cylinder) in the Seek mode.
38	LCT/DIR	O	Drive	**LOW CURRENT/DIRECTION:** Signals that the recording head has been positioned over the inner cylinders (44–77) of the floppy disk in the Read/Write mode. (The write current must be lowered when recording on the physically shorter inner cylinders of the disk. Most drives do not track the actual head position and require that the FDC supply this signal.) Determines the head step direction in the Seek mode. In the Seek mode, a high level on this pin steps the read/write head toward the spindle (step-in); a low level steps the head away from the spindle (step-out).
39	RW/SEEK	O	Drive	**READ, WRITE/SEEK MODE SELECTOR:** A high level selects the Seek mode; a low level selects the Read/Write mode.
40	V_{CC}			+5V DC Power.

*Disabled when CS is high.

Interface Registers

To support information transfer between the FDC and the system processor, the 8272 contains two 8-bit registers: the Main Status Register and the Data Register. The Main Status Register (read only) contains FDC status information and may be accessed at any time. The Main Status Register (Table 4) provides the system processor with the status of each disk drive, the status of the FDC, and the status of the processor interface. The Data Register (read/write) stores data, commands, parameters, and disk drive status information. The Data Register is used to program the FDC during the command phase and to obtain result information after completion of FDC operations. Data is read from, or written to, the FDC registers by the combination of the A0, $\overline{RD}$, $\overline{WR}$, and $\overline{CS}$ signals, as described in Table 3.

In addition to the Main Status Register, the FDC contains four additional status registers (ST0, ST1, ST2, and ST3). These registers are only available during the result phase of a command.

Table 3. FDC Read/Write Interface

$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	Function
0	0	0	1	Read Main Status Register
0	0	1	0	Illegal
0	0	0	0	Illegal
0	1	0	0	Illegal
0	1	0	1	Read from Data Register
0	1	1	0	Write into Data Register
1	X	X	X	Data Bus is three-stated

Table 4. Main Status Register Bit Definitions

Bit Number	Symbol	Description
0	D_0B	**DISK DRIVE 0 BUSY:** Disk Drive 0 is in the Seek mode.
1	D_1B	**DISK DRIVE 1 BUSY:** Disk Drive 1 is in the Seek mode.
2	D_2B	**DISK DRIVE 2 BUSY:** Disk Drive 2 is in the Seek mode.
3	D_3B	**DISK DRIVE 3 BUSY:** Disk Drive 3 is in the Seek mode.
4	CB	**FDC BUSY:** A read or write command is in process.
5	NDM	**NON-DMA MODE:** The FDC is in the non-DMA mode when this bit is high. This bit is set only during the execution phase of commands in the non-DMA mode. Transition to a low level indicates that the execution phase has ended.
6	DIO	**DATA INPUT/OUTPUT:** Indicates the direction of a data transfer between the FDC and the Data Register. When DIO is high, data is read from the Data Register by the processor; when DIO is low, data is written from the processor to the Data Register.
7	RQM	**REQUEST FOR MASTER:** Indicates that the Data Register is read to send data to, or receive data from, the processor.

Command/Result Phases

Table 5 lists the 8272 command set. For each of the fifteen commands, command and result phase data transfers are listed. A list of abbreviations used in the table is given in Table 6, and the contents of the result status registers (ST0–ST3) are illustrated in Table 7.

The bytes of data which are sent to the 8272 during the command phase, and are read out of the 8272 in the result phase, must occur in the order shown in Table 5. That is, the command code must be sent first and the other bytes sent in the prescribed sequence. All bytes of the command and result phases must be read/written as described. After the last byte of data in the command phase is sent to the 8272 the execution phase automatically starts. In a similar fashion, when the last byte of data is read from the 8272 in the result phase, the command is automatically ended and the 8272 is ready for a new command. A command may be aborted by simply raising the terminal count signal (pin 16). This is a convenient means of ensuring that the processor may always gain control of the 8272 (even if the disk system hangs up in an abnormal manner).

It is important to note that during the result phase all bytes shown in Table 5 must be read. The Read Data command, for example, has seven bytes of data in the result phase. All seven bytes must be read in order to successfully complete the Read Data command. The 8272 will not accept a new command until all seven bytes have been read. The number of command and result bytes varies from command-to-command.

In order to read data from, or write data to, the Data Register during the command and result phases, the system processor must examine the Main Status Register to determine if the Data Register is available. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command word may be written into the 8272. Many of the commands require multiple bytes, and as a result, the Main Status Register must be read prior to each byte transfer to the 8272. To read status bytes during the result phase, DIO and RQM in the Main Status Register must both be high. Note, checking the Main Status Register in this manner before each byte transfer to/from the 8272 is required only in the command and result phases, and is NOT required during the execution phase.

Execution Phase

All data transfers to (or from) the floppy drive occur during the execution phase. The 8272 has two primary modes of operation for data transfers (selected by the specify command):

1. DMA mode
2. non-DMA mode

In the DMA mode, DRQ (DMA Request) is activated for each transfers request. The DMA controller responds to DRQ with $\overline{\text{DACK}}$ (DMA Acknowledge) and $\overline{\text{RD}}$ (for read commands) or $\overline{\text{WR}}$ (for write commands). DRQ is reset by the FDC during the transfer. INT is activated after the last data transfer, indicating the completion of the execution phase, and the beginning of the result phase. In the DMA mode, the terminal count (TC/EOP) output of the DMA controller should be connected to the 8272 TC input to properly terminate disk data transfer commands.

Table 5. 8272 Command Set

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after command execution
	R				H					
	R				R					
	R				N					
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EC 1					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the main-system and the FDD
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system. FDC reads the complete track contents from the physical index mark to EOT
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the track is stored in Data Register
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information during Execution Phase
	R				H					
	R				R					
	R				N					
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				N					Bytes/Sector
	W				SC					Sectors/Track
	W				GPL					Gap 3
	W				D					Filter Byte
Execution										FDC formats an entire track
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					In this case, the ID information has no meaning
	R				H					
	R				R					
	R				N					
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data compared between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

207875-18

NOTE:
1. A_0 = 1 for all operations.

Table 5. 8272 Command Set (Continued)

PHASE	R/W	DATA BUS D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	REMARKS
SCAN LOW OR EQUAL			
Command	W	MT MFM SK 1 1 0 0 1	Command Codes
	W	0 0 0 0 0 HDS DS1 DS0	
	W	C	Sector ID information prior Command execution
	W	H	
	W	R	
	W	N	
	W	EOT	
	W	GPL	
	W	STP	
Execution			Data compared between the FDD and the main-system
Result	R	ST 0	Status information after Command execution
	R	ST 1	
	R	ST 2	
	R	C	Sector ID information after Command execution
	R	H	
	R	R	
	R	N	
SCAN HIGH OR EQUAL			
Command	W	MT MFM SK 1 1 1 0 1	Command Codes
	W	0 0 0 0 0 HDS DS1 DS0	
	W	C	Sector ID information prior Command execution
	W	H	
	W	R	
	W	N	
	W	EOT	
	W	GPL	
	W	STP	
Execution			Data compared between the FDD and the main-system
Result	R	ST 0	Status information after Command execution
	R	ST 1	
	R	ST 2	
	R	C	Sector ID information after Command execution
	R	H	
	R	R	
	R	N	

PHASE	R/W	DATA BUS D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	REMARKS
RECALIBRATE			
Command	W	0 0 0 0 0 1 1 1	Command Codes
	W	0 0 0 0 0 0 DS1 DS0	
Execution			Head retracted to Track 0
SENSE INTERRUPT STATUS			
Command	W	0 0 0 0 1 0 0 0	Command Codes
Result	R	ST 0	Status information at the end of each seek operation about the FDC
	R	C	
SPECIFY			
Command	W	0 0 0 0 0 0 1 1	Command Codes
	W	SPT ◄ HUT	Timer Settings
	W	HLT ► ND	
SENSE DRIVE STATUS			
Command	W	0 0 0 0 0 1 0 0	Command Codes
	W	0 0 0 0 0 HDS DS1 DS0	
Result	R	ST 3	Status information about the FDD
SEEK			
Command	W	0 0 0 0 1 1 1 1	Command Codes
	W	0 0 0 0 0 HDS DS1 DS0	
	W	C	
Execution			Head is positioned over proper Cylinder on Diskette
INVALID			
Command	W	Invalid Codes	Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R	ST 0	ST 0 = 80 (16)

207875-19

Table 6. Command/Result Parameter Abbreviations

Symbol	Description
C	**CYLINDER ADDRESS:** The currently selected cylinder address (0 to 76) on the disk.
D	**DATA PATTERN:** The pattern to be written in each sector data field during formatting.
DS0,DS1	**DISK DRIVE SELECT:**

DS1	DS0	
0	0	Drive 0
0	1	Drive 1
1	0	Drive 2
1	1	Drive 3

DTL	**SPECIAL SECTOR SIZE:** During the execution of disk read/write commands, this parameter is used to temporarily alter the effective disk sector size. By setting N to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the diskette) is larger than DTL specifies, the remainder of the actual sector is not passed to the system during read commands; during write commands, the remainder of the actual sector is written with all-zeroes bytes. DTL should be set to FF hexadecimal when N is not zero.
EOT	**END OF TRACK:** The final sector number of the current track.

Table 6. Command/Result Parameter Abbreviations (Continued)

Symbol	Description
GPL	**GAP LENGTH:** The gap 3 size. (Gap 3 is the space between sectors excluding the VCO synchronization field as defined in section 3.)
H	**HEAD ADDRESS:** Selected head: 0 or 1 (disk side 0 or 1, respectively) as encoded in the sector ID field.
HLT	**HEAD LOAD TIME:** Defines the time interval that the FDC waits after loading the head before initiating a read or write operation. Programmable from 2 to 254 milliseconds (in increments of 2 ms).
HUT	**HEAD UNLOAD TIME:** Defines the time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Programmable from 16 to 240 milliseconds (in increments of 16 ms).
MFM	**MFM/FM MODE SELECTOR:** Selects MFM double-density recording mode when high, FM single-density mode when low.
MT	**MUTLI-TRACK SELECTOR:** When set, this flag selects the multi-track operating mode. In this mode (used only with dual-sided disks), the FDC treats a complete cylinder (under both read/write head 0 and read/write head 1) as a single track. The FDC operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set (high), a multi-sector read operation will automatically continue to the first sector under head 1 when FDC finishes operating on the last sector under head 0.
N	**SECTOR SIZE:** The number of data bytes within a sector. (See Table 9.)
ND	**NON-DMA MODE FLAG:** When set (high), this flag indicates that the FDC is to operate in the non-DMA mode. In this mode, the processor is interrupted for each data transfer. When low, the FDC interfaces to a DMA controller by means of the DRQ and DACK signals.
R	**SECTOR ADDRESS:** Specifies the sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of first sector to be read or written.
SC	**NUMBER OF SECTORS PER TRACK:** Specifies the number of sectors per track to be initialized by the Format Track command.
SK	**SKIP FLAG:** When this flag is set, sectors containing deleted data address marks will automatically be skipped during the execution of multi-sector Read Data or Scan commands. In the same manner, a sector containing a data address mark will automatically be skipped during the execution of a multi-sector Read Delected Data command.
SRT	**STEP RATE INTERVAL:** Defines the time interval between step pulses issued by the FDC (track-to-track access time). Programmable from 1 to 16 milliseconds (in increments of 1 ms).

Table 6. Command/Result Parameter Abbreviations (Continued)

Symbol	Description
ST0,ST3 ST2,ST3	**STATUS REGISTER 0–3:** Registers within the FDC that store status information after a command has been executed. This status information is available to the processor during the Result Phase after command execution. These registers may only be read after a command has been executed (in the exact order shown in Table 5 for each command). These registers should not be confused with the Main Status Register.
STP	**SCAN SECTOR INCREMENT:** During Scan operations, this parameter is added to the current sector number in order to determine the next sector to be scanned.

Table 7. Status Register Definitions

Bit Number	Symbol	Description
Status Register 0		
7,6	IC	**INTERRUPT CODE:** 00— Normal termination of command. The specified command was properly executed and completed without error. 01— Abnormal termination of command. Command execution was started but could not be successfully completed. 10— Invalid command. The requested command could not be executed. 11— Abnormal termination. During command execution, the disk drive ready signal changed state.
5	SE	**SEEK END:** This flag is set (high) when the FDC has completed the Seek command and the read/write head is positioned over the correct cylinder.
4	EC	**EQUIPMENT CHECK ERROR:** This flag is set (high) if a fault signal is received from the disk drive or if the track 0 signal fails to become active after 77 step pulses (Recalibrate command).
3	NR	**NOT READY ERORR:** This flag is set if a read or write command is issued and either the drive is not ready or the command specifies side 1 (head 1) of a single-sided disk.
2	H	**HEAD ADDRESS:** The head address at the time of the interrupt.
1,0	DS1,DS0	**DRIVE SELECT:** The number of the drive selected at the time of the interrupt.
Status Register 1		
7	EN	**END OF TRACK ERROR:** This flag is set if the FDC attempts to access a sector beyond the final sector of the track.
6		Not used. This bit is always low.
5	DE	**DATA ERROR:** Set when the FDC detects a CRC error in either the ID field or the data field of a sector.

Table 7. Status Register Definitions (Continued)

Bit Number	Symbol	Description
Status Register 1 (Continued)		
4	OR	**OVERRUN ERROR:** Set (during data transfers) if the FDC does not receive DMA or processor service within the specified time interval.
3		Not used. This bit is always low.
2	ND	**SECTOR NOT FOUND ERROR:** This flag is set by any of the following conditions. a) The FDC cannot locate the sector specified in the Read Data, Read Delected Data, or Scan command. b) The FDC cannot locate the staring sector specified in the Read Track command. c) The FDC cannot read the ID field without error during a Read ID command.
1	NW	**WRITE PROTECT ERROR:** This flag is set if the FDC detects a write protect signal from the disk drive during the execution of a Write Data, Write Deleted Data, or Format Track command.
0	MA	**MISSING ADDRESS MARK ERROR:** This flag is set by either of the following conditions: a) The FDC cannot detect the ID address mark on the specified track (after two occurrences of the physical index mark). b) The FDC cannot detect the data address mark or deleted data address mark on the specified track. (See also the MD bit of Status Register 2.)
Status Register 2		
7		**NOT USED:** This bit is always low.
6	CM	**CONTROL MARK:** This flag is set when the FDC encounters one of the following conditions: a) A deleted data address mark during the execution of a Read Data or Scan command. b) A data address mark during the execution of a Read Deleted Data command.
5	DD	**DATA ERROR:** Set (high) when the FDC detects a CRC error in a sector data field. This flag is not set when a CRC eror is detected in the ID field.
4	WC	**CYLINDER ADDRESS ERROR:** Set when the cylinder address from the disk sector ID field is different from the current cylinder address maintained within the FDC.
3	SH	**SCAN HIT:** Set during the execution of the Scan command if the scan condition is satisfied.
2	SN	**SCAN NOT SATISFIED:** Set during execution of the Scan command if the FDC cannot locate a sector on the specified cylinder that satisfies the scan condition.
1	BC	**BAD TRACK ERROR:** Set when the cylinder address from the disk sector ID field is FF hexadecimal and this cylinder address is different from the current cylinder address maintained within the FDC. This all "ones" cylinder number indicates a bad track (one containing hard errors) according to the IBM soft-sectored format specifications.
0	MD	**MISSING DATA ADDRESS MARK ERROR:** Set if the FDC cannot detect a data address mark or deleted data address mark on the specified track.

Table 7. Status Register Definitions (Continued)

Bit Number	Symbol	Description
Status Register 3		
7	FT	**FAULT:** This flag indicates the status of the fault signal from the selected disk drive.
6	WP	**WRITE PROTECTED:** This flag indicates the status of the write protect signal from the selected disk drive.
5	RDY	**READY:** This flag indicates the status of the ready signal from the selected disk drive.
4	TO	**TRACK 0:** This flag indicates the status of the track 0 signal from the selected disk drive.
3	TS	**TWO-SIDED:** This flag indicates the status of the two-sided signal from the selected disk drive.
2	H	**HEAD ADDRESS:** This flag indicates the status of the side select signal for the currently selected disk drive.
1,0	DS1,DS0	**DRIVE SELECT:** Indicates the currently selected disk drive number.

In the non-DMA mode, transfers requests are indicated by activation of both the INT output signal and the RQM flag (bit 7) in the Main Status Register. INT can be used for interrupt-driven systems and RQM can be used for polled systems. The system processor must respond to the transfer request by reading data from (activating $\overline{RD}$), or writing data to (activating $\overline{WR}$), the FDC. This response removes the transfer request (INT and RQM are set inactive). After completing the last transfer, the 8272 activates the INT output to indicate the beginning of the result phase. In the non-DMA mode, the processor must activate the TC signal to the FDC (normally by means of an I/O port) after the transfer request for the last data byte has been received (by the processor) and before the appropriate data byte has been read from (or written to) the FDC.

In either mode of operation (DMA or non-DMA), the execution phase ends when a terminal count signal is sensed or when the last sector on a track (the EOT parameter—Table 5) has been read or written. In addition, if the disk drive is in a "not ready" state at the beginning of the execution phase, the "not ready" flag (bit 3 in Status Register 0) is set (high) and the command is terminated.

If a fault signal is received from the disk drive at the end of a write operation (Write Data, Write Deleted Data, or Format), the FDC sets the "equipment check" flag (bit 4 in Status Register 0), and terminates the command after setting the interrupt code (bits 7 and 6 of Status Register 0) to "01" (bit 7 low, bit 6 high).

Multi-Sector and Multi-Track Transfers

During disk read/write transfers (Read Data, Write Data, Read Deleted Data, and Write Deleted Data), the FDC will continue to transfer data from sequential sectors until the TC input is sensed. In the DMA mode, the TC input is normally connected to the TC/EOP (terminal count) output of the DMA controller. In the non-DMA mode, the processor directly controls the FDC TC input as previously described. Once the TC input is received, the FDC stops requesting data transfers (from the system processor or DMA controller). The FDC, however, continues to read data from, or write data to, the floppy disk until the end of the current disk sector. During a disk read operation, the data read from the disk (after reception of the TC input) is discarded, but the data CRC is checked for errors; during a disk write operation, the remainder of the sector is filled with all-zero bytes.

If the TC signal is not received before the last byte of the current sector has been transferred to/from the system, the FDC increments the sector number by one and initiates a read or write command for this new disk sector.

The FDC is also designed to operate in a multi-track mode for dual-sided disks. In the multi-track mode (specified by means of the MT flag in the command byte—Table 5) the FDC will automatically increment the head address (from 0 to 1) when the last sector (on the track under head 0) has been read or written. Reading or writing is then continued on the first sector (sector 1) of head 1.

Drive Status Polling

After the power-on reset, the 8272 automatically enters a drive status polling mode. If a change in drive status is detected (all drives are assumed to be "not ready" at power-on), an interrupt is generated. The 8272 continues this status polling between command executions (and between step pulses in the Seek command). In this manner, the 8272 automatically notifies the system processor when a floppy disk is inserted, removed, or changed by the operator.

Command Details

During the command phase, the Main Status Register must be polled by the CPU before each byte is written into the Data Register. The DI0 (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command may be written into the 8272. The beginning of the execution phase for any of these commands will cause DI0 to be set high and RQM to be set low.

The following paragraphs describe the fifteen FDC commands in detail.

Specify

The Specify command is used prior to performing any disk operations (including the formatting of a new disk) to define drive/FDC operating characteristics. The Specify command parameters set the values for three internal timers:

1. Head Load Time (HLT)—This seven-bit value defines the time interval that the FDC waits after loading the head before initiating a read or write operation. This timer is programmable from 2 to 254 milliseconds in increments of 2 ms.
2. Head Unload Time (HUT)—This four-bit value defines the time from the end of the execution phase (of a read or write command) until the head is unloaded. This timer is programmable from 16 to 240 milliseconds in increments of 16 ms. If the processor issues another command before the head unloads, the head will remain loaded and the head load wait will be eliminated.
3. Step Rate Time (SRT)—This four-bit value defines the time interval between step pulses issued by the FDC (track-to-track access time). This timer is programmable from 1 to 16 milliseconds in increments of 1 ms.

The time intervals mentioned above are a direct function of the FDC clock (CLK on pin 19). Times indicated above are for an 8 MHz clock.

The Specify command also indicates the choice of DMA or non-DMA operation (by means of the ND bit). When this bit is high the non-DMA mode is selected; when ND is low, the DMA mode is selected.

Sense Drive Status

This command may be used by the processor whenever it wishes to obtain the status of the disk drives. Status Register 3 (returned during the result phase) contains the drive status information as described in Table 7.

Sense Interrupt Status

An interrupt signal is generated by the FDC when one or more of the following events occurs:

1) The FDC enters the result phase for:
   a) Read Data command
   b) Read Track command
   c) Read ID command
   d) Read Deleted Data command
   e) Write Data command
   f) Format Track command
   g) Write Deleted Data command
   h) Scan commands
2) The ready signal from one of the disk drives changes state.
3) A Seek or Recalibrate command completes operation.
4) The FDC requires a data transfer during the execution phase of a command in the non-DMA mode.

Interrupts caused by reasons (1) and (4) above occur during normal command operations and are easily discernible by the processor. However, interrupts caused by reasons (2) and (3) above are uniquely identified with the aid of the Sense Interrupt Status command. This command, when issued, resets the interrupt signal and by means of bits 5, 6, and 7 of Status Register 0 (returned during the result phase) identifies the cause of the interrupt (see Table 8).

Table 8. Interrupt Codes

Seek End Bit 5	Interrupt Code		Cause
	Bit 6	Bit 7	
0	1	1	Read Line changed state, either polarity
1	0	0	Normal Termination of Seek or Recalibrate Command
1	1	0	Abnormal Termination of Seek or Recalibrate Command

Neither the Seek nor the Recalibrate command has a result phase. Therefore, it is mandatory to use the Sense Interrupt Status Command after these commands to effectively terminate them and to provide verification of the disk head position.

When an interrupt is received by the processor, the FDC busy flag (bit 4) and the non-DMA (bit 5) may be used to distinguish the above interrupt causes:

bit 5	bit 4	
0	0	Asynchronous event-(2) or (3) above
0	1	Result phase-(1) above
1	1	Data transfer required-(4) above

A single interrupt request to the processor may, in fact, be caused by more than one of the above events. The processor should continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are serviced.

Seek

The Seek command causes the drive's read/write head to be positioned over the specified cylinder. The FDC determines the difference between the current cylinder address and the desired (specified) address, and issues the appropriate number of step pulses. If the desired cylinder address is larger than the current address, the direction signal (LCT/DIR, pin 38) is set high (step-in); the direction signal is set low (step-out) if the desired cylinder address is less than the current address. No head movement occurs (no step pulses are issued) if the desired cylinder is the same as the current cylinder.

The rate at which step pulses are issued is controlled by the step rate time (SRT) in the Specify command. After each step pulse is issued, the desired cylinder address is compared against the current cylinder address. When the cylinder addresses are equal, the "seek end" flag (bit 5 in Status Register 0) is set (high) and the command is terminated. If the disk drive becomes "not ready" during the seek operation, the "not ready" flag (in Status Register 0) is set (high) and the command is terminated.

During the command phase of the Seek operation the FDC is in the FDC busy state, but during the execution phase it is in the non-busy state. While the FDC is in the non-busy state, another Seek command may be issued.. In this manner parallel seek operations may be in operation on up to four floppy disk drives at once. The Main Status Register contains a flag for each drive (Table 4) that indicates whether the associated drive is currently operating in the seek mode. When a drive has completed a seek operation, the FDC generates an interrupt. In response to this interrupt, the system software must issue a Sense Interrupt Status command. During the result phase of this command, Status Register 0 (containing the drive number in bits 0 and 1) is read by the processor.

Recalibrate

This command causes the read/write head of the disk drive to retract to the track 0 position. The FDC clears the contents of its internal cylinder counter, and checks the status of the track 0 signal from the disk drive. As long as the track 0 signal is low, the direction signal remains high and step pulses are issued. When the track 0 signal goes high, the seek end flag (in Status Register 0) is set (high) and the command is terminated. If the track 0 signal is still low after 77 step pulses have been issued, the seek end and equipment check flags (in Status Register 0) are both set and the Recalibrate command is terminated.

Recalibrate commands for multiple drives can be overlapped in the same manner that Seek commands are overlapped.

Format Track

The Format Track command formats or "initializes" a track on a floppy disk by writing the ID field, gaps, and address marks for each sector. Before issuing the Format command, the Seek command must be used to position the read/write head over the correct cylinder. In addition, a table of ID field values (cylinder, head, and sector addresses and sector length code) must be prepared before the command is executed. During command execution, the FDC accesses the table and, using the values supplied, writes each sector on the track. The ID field address mark originates from the FDC and is written automatically as the first byte of each sector's ID field. The cylinder, head, and sector addresses are taken, in order, from the table. The ID field CRC character (derived from the data written in the first five bytes) is written as the last two bytes of the ID field. Gaps are written automatically by the FDC, with the length of the variable gap determined by one of the Format command parameters.

The data field address mark is generated by the FDC and is written automatically as the first byte of the data field. The data pattern specified in the command phase is written into each data byte of each sector. A CRC character is derived from the data address mark and the data written in the sector's data field. The two CRC bytes are appended to the last data byte.

The formatting of a track begins at the physical index mark. As previously mentioned, the order of sector assignment is taken directly from the formatting table. Four entries are required for each sector: a cylinder address, a head address, a sector address, and a sector length code. The cylinder address in the ID field should be equal to the cylinder address of the track currently being formatted.

The sector addresses must be unique (no two equal). The order of the sector entries in the table is the sequence in which sector numbers appear on the track when it is formatted. The number of entry sets (cylinder, head, and sector address and sector length code) must equal the number of sectors allocated to the track (specified in the command phase).

Since the sector address is supplied, in order, for each sector, tracks can be formatted sequentially (the first sector following the index mark is assigned sector address 1, the adjacent sector is assigned sector address 2, and so on) or sector numbers can be interleaved (see section 3) on a track.

Table 9 lists recommended gap sizes and sectors/track for various sector sizes.

Read Data

Nine (9) bytes are required to complete the command phase specification for the Read Data command. During the execution phase, the FDC loads the head (if it is in the unloaded state), waits the specified head load time (defined in the Specify command), and begins reading ID address marks and ID fields. When the requested sector address compares with the sector address read from the disk, the FDC outputs data (from the data field) byte-by-byte to the system. The Read Data command automatically operates in the multi-sector mode described earlier. In addition, multi-track operation may be specified by means of the MT command flag (Table 5). The amount of data that can be transferred with a single command to the FDC depends on the multi-track flag, the recording density flag, and the number of bytes per sector.

During the execution of read and write commands, the special sector size parameter (DTL) is used to temporarily alter the effective disk sector size. By setting the sector size code (N) to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the disk) is larger than DTL specifies, only the number of bytes specified by the DTL parameter are passed to the system; the remainder of the actual disk sector is not transferred (although the data is checked for CRC errors). Multi-sector read operations are performed in the same manner as they are when the sector size code is non-zero. (The N and DTL parameters are always present in the command sequence. DTL should be set to FF hexadecimal when N is not zero.)

If the FDC detects the physical index mark twice without finding the requested sector, the FDC sets the "sector not found error" flag (bit 2 in Status Register 1) and terminates the Read Data command. The interrupt code (bits 7 and 6 of Status Register 0) is set to "01." Note that the FDC searches for each sector in a multi-sector operation. Therefore, a "sector not found"' error may occur after successful transfer of one or more preceding sectors. This error could occur if a particular sector number was not included when the track was first formatted or if a hard error on the disk has invalidated a sector ID field.

After reading the ID field and data field in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in the ID field), the FDC sets the "data error" flag in Status Register 1; if a CRC error occurs in the data field, the FDC sets the "data error" flag in Status Register 2. In either error condition, the FDC terminates the Read Data command. The interrupt code (bits 7 and 6 in Status Register 0) is set to "01."

If the FDC reads a deleted data address mark from the disk, and the skip flag (specified during the command phase) is not set, the FDC sets the "control mark" flag (bit 6 in Status Register 2) and terminates the Read Data command (after reading all the data in the sector). If the skip flag is set, the FDC skips the sector with the deleted data address mark and reads the next sector. Thus, the skip flag may be used to cause the FDC to ignore deleted data sectors during a multi-sector read operation.

Table 9. Sector Size Relationships

Format	Sector Size	N Sector Size Code	SC Sectors/ Track	GPL(1) Gap 3 Length	GPL(2) Gap 3 Length	Remarks
FM Mode	128 bytes/Sector	00	$1A_{(16)}$	$07_{(16)}$	$1B_{(16)}$	IBM Diskette 1
	256	01	$0F_{(16)}$	$0E_{(16)}$	$2A_{(16)}$	IBM Diskette 2
	512	02	08	$1B_{(16)}$	$3A_{(16)}$	
MFM Mode	256	01	$1A_{(16)}$	$0E_{(16)}$	$36_{(16)}$	iBM Diskette 2D
	512	02	$0F_{(16)}$	$1B_{(16)}$	$54_{(16)}$	
	1024	03	08	$35_{(16)}$	$74_{(16)}$	IBM Diskette 2D

NOTES:
1. Suggested values of GPL in Read or Write commands to avoid splice point between data field and ID field of contiguous sectors.
2. Suggested values of GPL in Format command.

During disk data transfers between the FDC and the system, the FDC must be serviced by the system (processor or DMA controller) every 27 μs in the FM mode, and every 13 μs in the MFM mode. If the FDC is not serviced within this interval, the "overrun error" flag (bit 4 in Status Register 1) is set and the Read Data command is terminated.

If the processor terminates a read (or write) operation in the FDC, the ID information in the result phase is dependent upon the state of the multi-track flag and end of track byte. Table 11 shows the values for C, H, R, and N, when the processor terminates the command.

Write Data

Nine (9) bytes are required to complete the command phase specification for the Write Data command. During the execution phase the FDC loads the head (if it is in the unloaded state), waits the specified head load time (defined by the Specify command), and begins reading sector ID fields. When the requested sector address compares with the sector address read from the disk, the FDC reads data from the processor one byte at a time via the data bus and outputs the data to the data field of that sector. The CRC is computed on this data and two CRC bytes are written at the end of the data field.

The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (incorrect CRC) in one of the ID fields, it sets the "data error" flag (bit 5 in Status Register 1) and terminates the Write Data command. The interrupt code (bits 7 and 6 in Status Register 0) is set to "01."

The Write Data command operates in much the same manner as the Read Data command. The following items are the same; refer to the Read Data command for details:

- Multi-sector and Multi-track operation
- Data transfer capacity
- "End of track error" flag
- "Sector not found error" flag
- "Data error" flag
- Head unload time interval
- ID information when the processor terminates the command (see Table 11)
- Definition of DTL when N = 0 and when N ≠ 0

During the Write Data execution phase, data transfers between the processor and FDC must occur every 31 μs in the FM mode, and every 15 μs in the MFM mode. If the time interval between data transfers is longer than this, the FDC sets the "overrun error" flag (bit 4 in Status Register 1) and terminates the Write Data command.

Read Deleted Data

This command operates in almost the same manner as the Read Data command operates. The only difference involves the treatment of the data address mark and the skip flag. When the FDC detects a data address mark at the beginning of a data field (and the skip flag is not set), the FDC reads all the data in the sector, sets the "control mark" flag (bit 6 in Status Register 2), and terminates the command. If the skip flag is set, the FDC skips the sector with the data address mark and continues reading at the next sector. Thus, the skip flag may be used to cause the FDC to read only deleted data sectors during a multi-sector read operation.

Write Deleted Data

This command operates in the same manner as the Write Data command operates except that a deleted data address mark is written at the beginning of the data field instead of the normal data address mark. This command is used to mark a bad sector (containing a hard error) on the floppy disk.

Read Track

The Read Track command is similar to the Read Data command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering the physical index mark, the FDC starts reading all data fields on the track as continuous blocks of data. If the FDC finds an error in the ID field or data field CRC check bytes, it continues to read data from the track. The FDC compares the ID information read from each sector with the values specified during the command phase. If the specified ID field information is not found on the track, the "sector not found error" flag (in Status Register 1) is set. Multitrack and skip operations are not allowed with this command.

This command terminates when the last sector on the track has been read. (The number of sectors on the track is specified by the end of track parameter byte during the command phase.) If the FDC does not find an ID address mark on the disk after it encounters the physical index mark for the second time, it sets the "missing address mark error" flag (bit 0 in Status Register 1) and terminates the command. The interrupt code (bits 7 and 6 of Status Register 0) is set to "01."

Read ID

The Read ID command transfers (reads) the first correct ID field from the current disk track (following the physical index mark) to the processor. If no correct ID address mark is found on the track, the "missing address mark error" flag is set (bit 0 in Status Register 1). If no data mark is found on the track, the "sector not found error" flag is also set (bit 2 in Status Register 1). Either error condition causes the command to be terminated.

Scan Commands

The Scan commands allow the data being read from the disk to be compared against data supplied by the system (by the processor in non-DMA mode, and by the DMA controller in DMA mode). The FDC compares the data on a byte-by-byte basis, and searches for a sector of data that meets the conditions of "disk data equal to system data", "disk data less than or equal to system data", or "disk data greater than or equal to system data". Simple binary (ones complement) arithmetic is used for comparison (FF = largest number, 00 = smallest number). If, after a complete sector of data is compared, the conditions are not met, the sector number is incremented by the scan sector increment (specified in the command phase), and the scan operation is continued. The scan operation continues until one of the following conditions occurs; the conditions for scan are met (equal, low, or high), the last sector on the track is reached, or the terminal count signal is received.

If the conditions for scan are met, the FDC sets the "scan hit" flag (bit 3 in Status Register 2) and terminates the Scan command. If the conditions for scan are not met between the starting sector and the last sector on the track (specified in the command phase), the FDC sets the "scan not satisfied" flag (bit 2 in Status Register 2) and terminates the Scan command. The receipt of a terminal count signal from the processor or DMA controller during the scan operation will cause the FDC to complete the comparison of the particular byte which is in process, and to termiante the command. Table 10 shows the status of the "scan hit" and "scan not satisfied" flags under various scan termination conditions.

If the FDC encounters a deleted data address mark in one of the sectors and the skip flag is low, it regards the sector as the last sector on the cylinder, sets the "control mark" flag (bit 6 in Status Register 2) and terminates the command. If the skip flag is high, the FDC skips the sector with the deleted address mark, and reads the next sector. In this case, the FDC also sets the

Table 10. Scan Status Codes

Command	Status Register 2		Comments
	Bit 2 = SN	Bit 3 = SH	
Scan Equal	0	1	$D_{FDD} = D_{Processor}$
	1	0	$D_{FDD} \neq D_{Processor}$
Scan Low or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} < D_{Processor}$
	1	0	$D_{FDD} > D_{Processor}$
Scan High or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} > D_{Processor}$
	1	0	$D_{FDD} < D_{Processor}$

'control mark" flag (bit 6 in Stauts Register 2) in order to show that a deleted sector had been encountered.

NOTE:
During scan command execution, the last sector on the track must be read for the command to terminate properly. For example if the scan sector increment is set to 2, the end of track parameter is set to 26, and the scan begins at sector 21, sectors 21, 23, and 25 will be scanned. The next sector, 27 will not be found on the track and an abnormal command termination will occur. The command would be completed in a normal manner if either a) the scan had started at sector 20 or b) the end of track parameter had been set to 25.

During the Scan command, data is supplied by the processor or DMA controller for comparison against the data read from the disk. In order to avoid having the "overrun error" flag set (bit 4 in Status Register 1), it is necessary to have the data available in less than 27 μs (FM Mode) or 13 μs (MFM Mode). If an overrun error occurs, the FDC terminates the command.

Invalid Commands

If an invalid (undefined) command is sent to the FDC, the FDC will terminate the command. No interrupt is generated by the 8272 during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both set indicating to the processor that the 8272 is in the result phase and the contents of Status Register 0 must be read. When the processor reads Status Register 0 it will find an 80H code indicating that an invalid command was received.

A Sense Interrupt Status command must be sent after a Seek or Recalibrate interrupt; otherwise the FDC will consider the next command to be an invalid command. Also, when the last "hidden" interrupt has been serviced, further Sense Interrupt Status commands will result in invalid command codes.

Table 11. ID Information When Processor Terminates Command

MT	EOT	Final Sector Transferred to Processor	ID Information at Result Phase			
			C	H	R	N
0	1A	Sector 1 to 25 at Side 0	NC	NC	R + 1	NC
	0F	Sector 1 to 14 at Side 0				
	08	Sector 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	C + 1	NC	R = 01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sector 1 to 25 at Side 1	NC	NC	R + 1	NC
	0F	Sector 1 to 14 at Side 1				
	08	Sector 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C + 1	NC	R = 01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				
1	1A	Sector 1 to 25 at Side 0	NC	NC	R + 1	NC
	0F	Sector 1 to 14 at Side 0				
	08	Sector 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	NC	$\overline{\text{LSB}}$	R = 01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sector 1 to 25 at Side 1	NC	NC	R + 1	NC
	0F	Sector 1 to 14 at Side 1				
	08	Sector 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C + 1	$\overline{\text{LSB}}$	R = 01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				

NOTES:
1. NC (No Change): The same value as the one at the beginning of command execution.
2. $\overline{\text{LSB}}$ (Least Significant Bit): The least significant bit of H is complemented.

In some applications the user may wish to use this command as a No-Op command to place the FDC in a stand-by or no operation state.

5.0 THE DATA SEPARATOR

As briefly discussed in Section 2, LSI disk controllers such as the 8272 require external circuitry to generate a data window signal. This signal is used within the FDC to isolate the data bits contained within the READ DATA input signal from the disk drive. (The disk READ DATA signal is a composite signal constructed from both clock and data information.) After isolating the data bits from this input signal, the FDC assembles the data bits into 8-bit bytes for transfer to the system processor or memory.

Single Density

In a single-density (FM) recording (Figure 3), the bit cell is 4 microseconds wide. Each bit cell contains a clock bit at the leading edge of the cell. The data bit (if present) is always located at the center of the cell. The job of data separation is relatively straightforward for single-density; simply generate a data window 2 μs wide starting 1 μs after each clock bit. Since every cell has a clock bit, a fixed window reference is available for every data bit and because the window is 2 μs wide, a slightly shifted data bit will still remain within the data window.

A single-density data separator with these specifications may be easily generated using a digital or analog one-shot triggered by the clock bit.

Double-Density

Double-density (MFM) bit cells are reduced to 2 μs (in order to double the disk data storage capacity). Clock bits are inserted into the data stream only if data bits are not present in both the current and preceding bit cells (Figure 3). The data bit (if present) still occurs at the center of the bit cell and the clock bit (if present) still occurs at the leading edge of the bit cell.

MFM data separation has two problems. First, only some bit cells contain a clock bit. In this manner, MFM encoding loses the fixed bit cell reference pulse present in FM encoding. Second, the bit cell for MFM is one-half the size of the bit cell for FM. This shorter bit cell means that MRM cannot tolerate as large a playback data-shift (as FM can tolerate) without errors.

Since most playback data-shift is predictable, the FDC can precompensate the write data stream so that data/clock pulses will be correctly positioned for subsequent playback. This function is completely controlled by the FDC and is only required for MFM recording. During write operations, the FDC specifies an early, normal, or late bit positioning. This timing information is specified with respect to the FDC write clock. Early and late timing is typically 125 ns to 250 ns before or after the write clock transition (depending on disk drive requirements).

The data separator circuitry for double-density recording must continuously analyze the total READ DATA stream, synchronizing its operation (window generation) with the actual clock/data bits of the data stream. The data separation circuit must track the disk input data frequency very closely—unpredictable bit shifts leave less than 50 ns margin to the window edges.

Phase-Locked Loop

Only an analog phase-locked loop (PLL) can provide the reliability required for a double-density data separation circuit. (A phase-locked loop is an electronic circuit that constantly analyzes the frequency of an input signal and locks another oscillator to that frequency.) Using analog PLL techniques, a data separator can be designed with ± 1 ns resolution (this would require a 100 MHz clock in a digital phase-locked loop). The analog PLL determines the clock and data bit positions by sampling each bit in the serial data stream. The phase relationship between a data bit and the PLL generated data window is constantly fed back to adjust the position of the data window, enabling the PLL to track input data frequency changes, and thereby reliably read previously recorded data from a floppy disk.

PLL Design

A block diagram of the phase-locked loop described in this application note is shown in Figure 7. Basically, the phase-locked loop operates by comparing the frequency of the input data (from the disk drive) against the frequency of a local oscillator. The difference of these frequencies is used to increase or decrease the frequency of the local oscillator in order to bring its frequency closer to that of the input. The PLL synchronizes the local

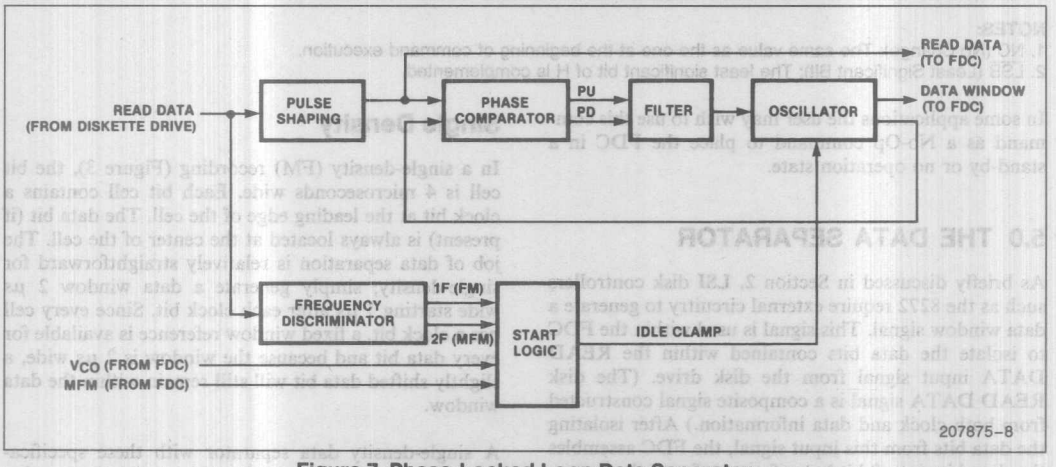


Figure 7. Phase-Locked Loop Data Separator

oscillator to the frequency of the input during the all "zeroes" synchronization field on the floppy disk (immediately preceding both the ID field and the data field.

The PLL consists of nine ICs and is located on page 3 of the schematics in the Appendix. The 8272 VCO output essentially turns the PLL circuitry on and off. When the PLL is off, it "idles" at its center frequency. The VCO turns the PLL on only when valid data is being received from the disk drive. The VCO turns the PLL on after the read/write head has been loaded and the head load time has elapsed. The PLL is turned off in the gap between the ID field and the data field and in the gap after the data field (before the next sector ID field). The GPL parameter in the FDC read and write commands specifies the elapsed time (number of data bytes) that the PLL is turned off in order to blank out discontinuities that appear in the gaps when the write current is turned on and off. The PLL operates with either MFM or FM input data. The MFM output from the FDC controls the PLL operation frequency.

The PLL consists of six functional blocks as follows:

1) Pulse Shaping—A 96LS02 senses a READ DATA pulse and provides a clean output signal to the FDC and to the PLL Phase Comparator and Frequency Discriminator circuitry.
2) Phase Comparator—The phase difference between the PLL oscillator and the READ DATA input is compared. Pump up (PU) and pump down (PD) error signals are derived from this phase difference and output to the filter. If there is no phase difference between the PLL oscillator and the READ DATA input, the PU and PD pulse widths are equal. If the READ DATA pulse occurs early, the PU duration is shorter than the PD duration. If the data pulse occurs late, the PU duration is longer than the PD duration.
3) Filter—This analog circuit filters the PU and PD pulses into an error voltage. This error voltage is buffered by an LM358 operational amplifier.
4. PLL Oscillator—This oscillator is composed of a 74LS393, 74LS74, and 96LS02. The oscillator frequency is controlled by the error voltage output by the filter. This oscillator also generates the data window signal to the FDC.
5. Frequency Discriminator—This logic tracks the READ DATA input from the disk drive and discriminates between the synchronization gap for FM recording (250 KHz) and the gap for MFM recording (500 KHz). Synchronization gaps immediately precede address marks.
6. Start Logic—The function of this logic is to clamp the PLL oscillator to its center frequency (2 MHz) until the FDC VCO signal is enabled and a valid data pattern is sensed by the frequency discriminator. The start logic (consisting of a 74LS393 and 74LS74) ensures that the PLL oscillator is started with zero phase error.

PLL Adjustments

The PLL must be initially adjusted to operate at its center frequency with the VCO output off and the adjustment jumper removed. The 5K timepot should be adjusted until the frequency at the test point (Q output of the 96LS02) is 2 MHz. The jumper should then be replaced for normal operation.

PLL Design Details

The following paragraphs describe the operational and design details of the phase-locked loop data separator illustrated in the appendix. Note that the analog section is operated from a separately filtered +5V supply.

Initialization

As long as the 8272 maintains a low VCO signal, the data separator logic is "turned off". In this state, the PLL oscillator (96LS02) is not oscillating and therefore the 2XBR signal is constantly low. In addition, the pump up (PU) and pump down (PD) signals are inactive (PU low and PD high), the CNT8 signal is inactive (low), and the filter input voltage is held at 2.5 volts by two 1 MΩ resistors between ground and +5 volts.

Floppy Disk Data

The data separator freqeuncy discriminator, the input pulse shaping circuitry, and the start logic are always enabled and respond to rising edges of the READ DATA signal. The rising edge of every data bit from the disk drive triggers two pulse shaping one-shots. The first pulse shaper generates a stable and well-defined 200 ns read data pulse for input to the 8272 and other portions of the data separator logic. The second one-shot generates a 2.5 μs data pulse that is used for input data frequency discrimination.

The frequency discriminator operates as illustrated in Figure 8. The 2F output signal is active (high) during reception of valid MFM (double-density) sync fields on the disk while the 1F signal is active (high) during reception of valid FM (single-density) sync fields. A multiplexer (controlled by the 8272 MFM signal) selects the appropriate 1F or 2F signal depending on the programmed mode.

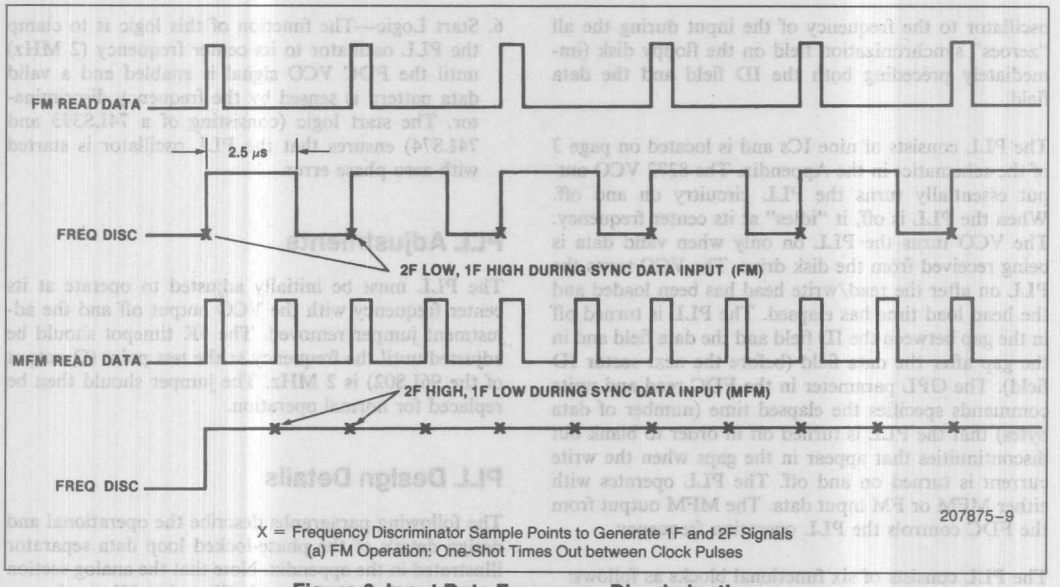


Figure 8. Input Data Frequency Discrimination

Startup

The data separator is designed to require reception of eight valid sync bits (one sync byte) before enabling the PLL oscillator and attempting to synchronize with the input data stream (see Figure 9). This delay ensures that the PLL will not erroneously synchronize outside a valid sync field in the data stream if the VCO signal is enabled slightly early. The sync bit counter is asynchronously reset by the CNTEN signal when valid sync data is not being recevied by the drive.

Once the VCO signal is active and eight sync bits have been counted, the CNT8 signal is enabled. This signal turns on the PLL oscillator. Note that this oscillator starts synchronously with the rising edge of the disk input data (becasue CNT8 is synchronous with the data rising edge) and the oscillator also starts at its center frequency of 2 MHz (because the LM348 filter input is held at its center voltage of approximately 2.5 volts). This frequency is divided by two and four to generate the 2XBR signal (1 MHz for MFM and 500 KHz for FM).

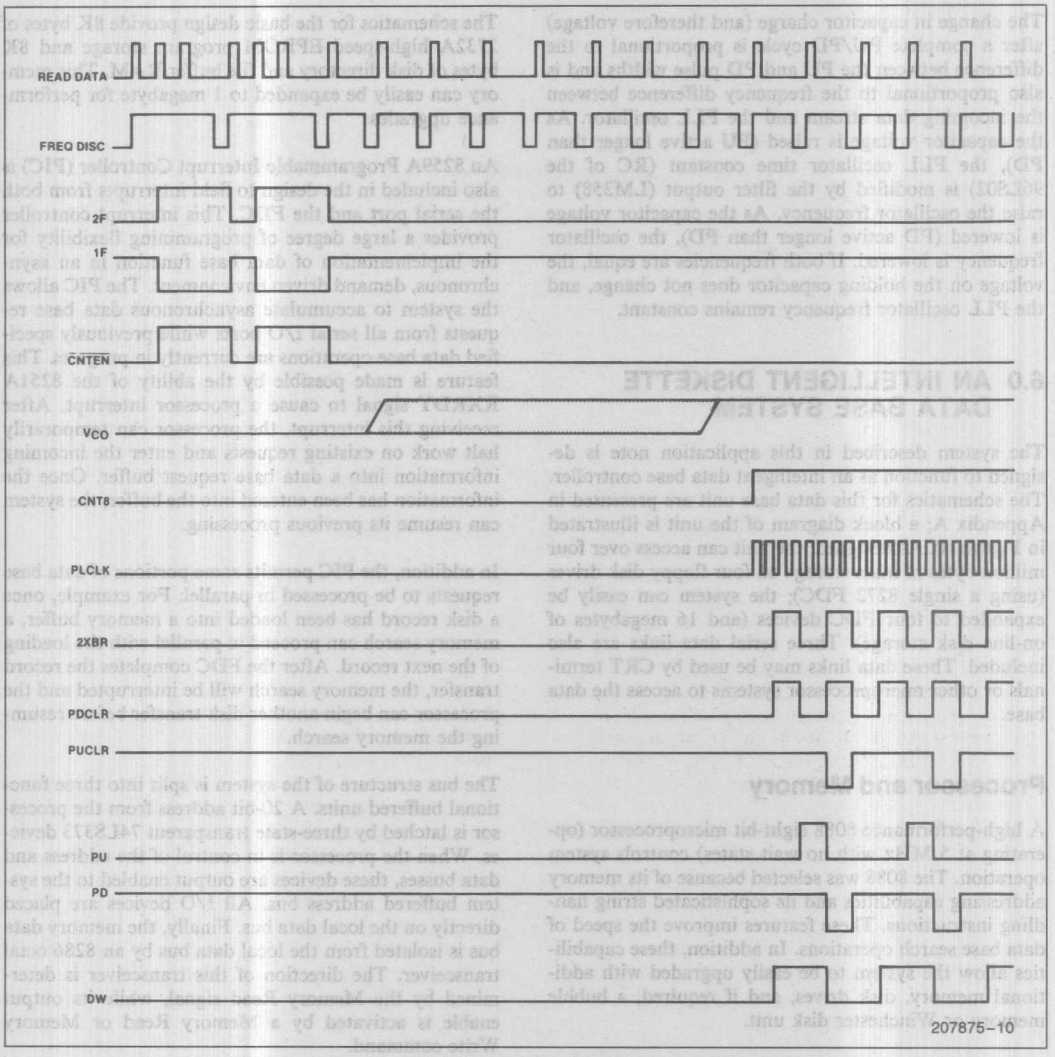


Figure 9. Typical Data Separator Startup Timing Diagram

PLL Synchronization

At this point, the PLL is enabled and begins to synchronize with the input data stream. This synchronization is accomplished very simply in the following manner. The pump up (PU) signal is enabled on the rising edge of the READ DATA from the disk drive. (When the PLL is synchronized with the data stream, this point will occur at the same time as the falling edge of the 2XBR signal as shown in Figure 9). The PU Signal is turned off and the PD signal is activated on the next rising edge of the 2XBR clock. With this scheme, the difference between PU active time and the PD active time is equal to the difference between the input bit rate and the PLL clock rate. Thus, if PU is turned on longer than PD is on, the input bit rate is faster than the PLL clock.

As long as PU and PD are both inactive, no charge is transferred to or from the LM358 input holding capacitor, and the PLL output frequency is maintained (the LM358 operational amplifier has a very high input impedance). Whenver PU is turned on, current flows from the +5 volt supply through a 20K resistor into the holding capacitor. When the PD signal is turned on, current flows from the holding capacitor to ground through a 20K resistor. In this manner, both the pump up and pump down charging rates are balanced.

The change in capacitor charge (and therefore voltage) after a complete PU/PD cycle is proportional to the difference between the PU and PD pulse widths and is also proportional to the frequency difference between the incoming data stream and the PLL oscillator. As the capacitor voltage is raised (PU active longer than PD), the PLL oscillator time constant (RC of the 96LS02) is modified by the filter output (LM358) to raise the oscillator frequency. As the capacitor voltage is lowered (PD active longer than PD), the oscillator frequency is lowered. If both frequencies are equal, the voltage on the holding capacitor does not change, and the PLL oscillator frequency remains constant.

6.0 AN INTELLIGENT DISKETTE DATA BASE SYSTEM

The system described in this application note is designed to function as an intelligent data base controller. The schematics for this data base unit are presented in Appendix A; a block diagram of the unit is illustrated in Figure 10. As designed, the unit can access over four million bytes of mass storage on four floppy disk drives (using a single 8272 FDC); the system can easily be expanded to four FDC devices (and 16 megabytes of on-line disk storage). Three serial data links are also included. These data links may be used by CRT terminals or other microprocessor systems to access the data base.

Processor and Memory

A high-performance 8088 eight-bit microprocessor (operating at 5 MHz with no wait states) controls system operation. The 8088 was selected because of its memory addressing capabilities and its sophisticated string handling instructions. These features improve the speed of data base search operations. In addition, these capabilities allow the system to be easily upgraded with additional memory, disk drives, and if required, a bubble memory or Winchester disk unit.

The schematics for the basic design provide 8K bytes of 2732A high-speed EPROM program storage and 8K bytes of disk directory and file buffer RAM. This memory can easily be expanded to 1 megabyte for performance upgrades.

An 8259A Programmable Interrupt Controller (PIC) is also included in the design to field interrupts from both the serial port and the FDC. This interrupt controller provides a large degree of programming flexibility for the implementation of data base function in an asynchronous, demand driven environment. The PIC allows the system to accumulate asynchronous data base requests from all serial I/O ports while previously specified data base operations are currently in progress. This feature is made possible by the ability of the 8251A RXRDY signal to cause a processor interrupt. After receiving this interrupt, the processor can temporarily halt work on existing requests and enter the incoming information into a data base request buffer. Once the information has been entered into the buffer, the system can resume its previous processing.

In addition, the PIC permits some portions of data base requests to be processed in parallel. For example, once a disk record has been loaded into a memory buffer, a memory search can proceed in parallel with the loading of the next record. After the FDC completes the record transfer, the memory search will be interrupted and the processor can begin another disk transfer before resuming the memory search.

The bus structure of the system is split into three functional buffered units. A 20-bit address from the processor is latched by three-state transparent 74LS373 devices. When the processor is in control of the address and data busses, these devices are output enabled to the system buffered address bus. All I/O devices are placed directly on the local data bus. Finally, the memory data bus is isolated from the local data bus by an 8286 octal transceiver. The direction of this transceiver is determined by the Memory Read signal, while its output enable is activated by a Memory Read or Memory Write command.

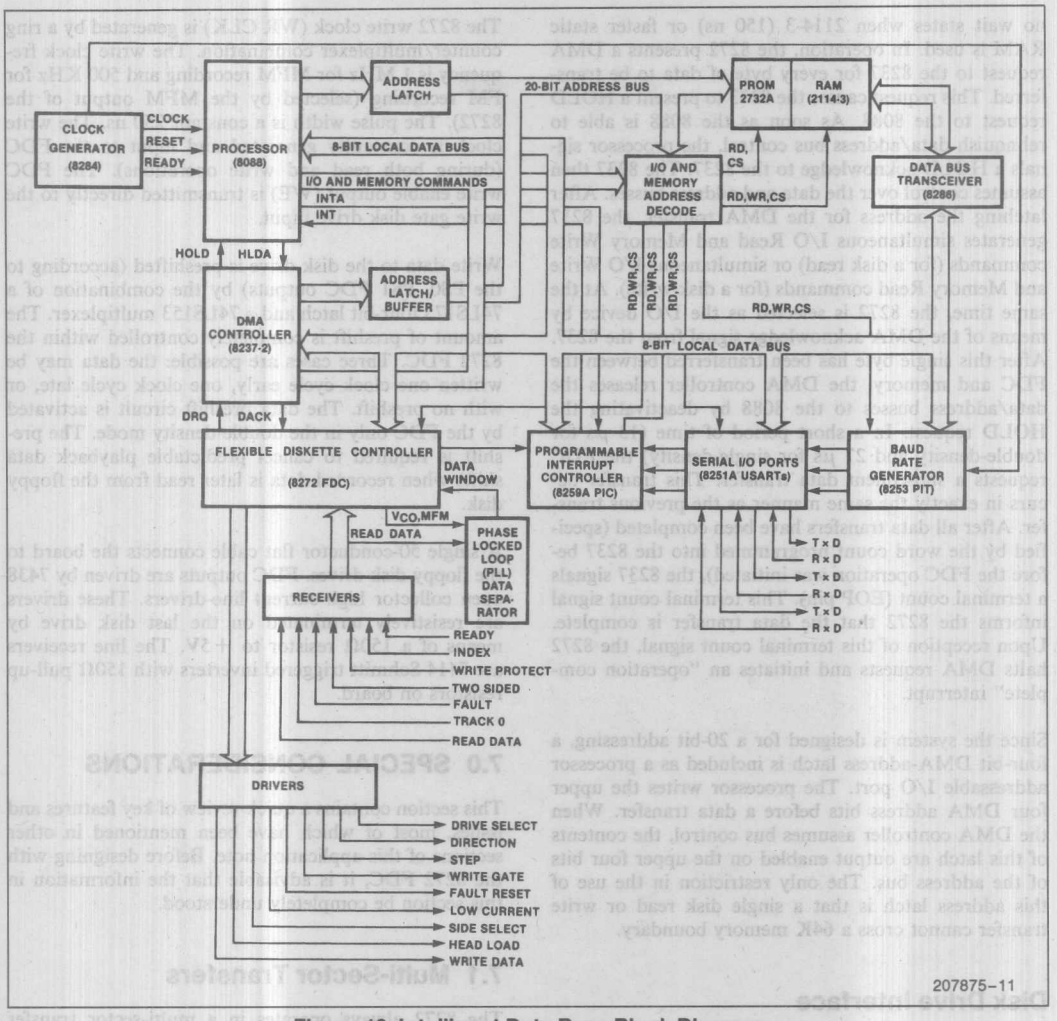


Figure 10. Intelligent Data Base Block Diagram

Serial I/O

The three RS-232-C compatible serial I/O ports operate at software-programmable baud rates to 19.2K. Each I/O port is controlled by an 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter). Each USART is individually programmable for operation in many synchronous and asynchronous serial data transmission formats (including IBM Bi-sync). In operation, USART error detection circuits can check for parity, data overrun, and framing errors. An 8253 Programmable Interval Timer is employed to generate the baud rates for the serial I/O ports.

The Transmitter Read and Receiver Ready output signals of the 8251As are routed to the interrupt inputs of the 8259A interrupt controller. These signals interrupt processor execution when a data byte is received by a USART and also when the USART is ready to accept another data byte for transmission.

DMA

The 8272 FDC interfaces to system memory by means of an 8237-2 high-speed DMA controller. Transfers between the disk controller and memory also operate with

no wait states when 2114-3 (150 ns) or faster static RAM is used. In operation, the 8272 presents a DMA request to the 8237 for every byte of data to be transferred. This request causes the 8273 to present a HOLD request to the 8088. As soon as the 8088 is able to relinquish data/address bus control, the processor signals a HOLD acknowledge to the 8237. The 8237 then assumes control over the data and address busses. After latching the address for the DMA transfer, the 8237 generates simultaneous I/O Read and Memory Write commands (for a disk read) or simultaneous I/O Write and Memory Read commands (for a disk write). At the same time, the 8272 is selected as the I/O device by means of the DMA acknowledge signal from the 8237. After this single byte has been transferred between the FDC and memory, the DMA controller releases the data/address busses to the 8088 by deactivating the HOLD request. In a short period of time (13 μs for double-density and 27 μs for single-density) the FDC requests a subsequent data transfer. This transfer occurs in exactly the same manner as the previous transfer. After all data transfers have been completed (specified by the word count programmed into the 8237 before the FDC operation was initiated), the 8237 signals a terminal count (EOP pin). This terminal count signal informs the 8272 that the data transfer is complete. Upon reception of this terminal count signal, the 8272 halts DMA requests and initiates an "operation complete" interrupt.

Since the system is designed for a 20-bit addressing, a four-bit DMA-address latch is included as a processor addressable I/O port. The processor writes the upper four DMA address bits before a data transfer. When the DMA controller assumes bus control, the contents of this latch are output enabled on the upper four bits of the address bus. The only restriction in the use of this address latch is that a single disk read or write transfer cannot cross a 64K memory boundary.

Disk Drive Interface

The 8272 FDC may be interfaced to a maximum of four eight-inch floppy disk drives. Both single- and double-density drives are accommodated using the data separation circuit described in section 5. In addition, single- or dual-sided disk drives may be used. The 8272 is driven by an 8 MHz crystal controller clock produced by an 8224 clock generator.

Drive select signals are decoded by means of a 74LS139 from the DS0, DS1 outputs of the FDC. The fault reset, step, low current, and direction outputs to the disk drives are generated from the FR/STEP, LCT/DIR, and RW/SEEK FDC output signals by means of a 74LS240. The other half of the 74LS240 functions as an input multiplexer for the disk write protect, two-sided, fault, and track zero status signals. These signals are multiplexed into the WP/TS and FLT/TRK0 inputs to the 8272.

The 8272 write clock (WR CLK) is generated by a ring counter/multiplexer combination. The write clock frequency is 1 MHz for MFM recording and 500 KHz for FM recording (selected by the MFM output of the 8272). The pulse width is a constant 250 ns. The write clock is constantly generated and input to the FDC (during both read and write operations). The FDC write enable output (WE) is transmitted directly to the write gate disk drive input.

Write data to the disk drive is preshifted (according to the PS0, PS1 FDC outputs) by the combination of a 74LS175 four-bit latch and a 74LS153 multiplexer. The amount of preshift is completely controlled within the 8272 FDC. Three cases are possible: the data may be written one clock cycle early, one clock cycle late, or with no preshift. The data preshift circuit is activated by the FDC only in the double-density mode. The preshift is required to cancel predictable playback data shifts when recorded data is later read from the floppy disk.

A single 50-conductor flat cable connects the board to the floppy disk drives. FDC outputs are driven by 7438 open collector high-current line-drivers. These drivers are resistively terminated on the last disk drive by means of a 150Ω resistor to +5V. The line receivers are 7414 Schmitt triggered inverters with 150Ω pull-up resistors on board.

7.0 SPECIAL CONSIDERATIONS

This section contains a quick review of key features and issues, most of which have been mentioned in other sections of this application note. Before designing with the 8272 FDC, it is advisable that the information in this section be completely understood.

7.1 Multi-Sector Transfers

The 8272 always operates in a multi-sector transfer mode. The 8272 continues to transfer data until the TC input is activated. In a DMA configuration, the TC input of the 8272 must always be connected to the EOP/TC output of the DMA controller. When multiple DMA channels are used on a single DMA controller, EOP must be gated with the select signal for the proper FDC. If the TCD signal is not gated, a terminal count on another channel will abort FDC operation.

In a processor driven configuration with no DMA controller, the system must count the transfers and supply a TC signal to the FDC. In a DMA environment, OR-ing a programmable TC with the TC from the DMA controller is a convenient means of ensuring that the processor may always gain control of the FDC (even if the diskette system hangs up in an abnormal manner).

7.2 Processor Command/Result Phase Interface

In the command phase, the processor must write the exact number of parameters in the exact order shown in Table 5. During the result phase, the processor must read the complete result status. For example, the Format Track command requires six command bytes and presents seven result bytes. The 8272 will not accept a new command until all result bytes are read. Note that the number of command and result bytes varies from command-to-command. Command and result phases cannot be shortened.

During both the command and result phases, the Main Status Register must be read by the processor before each byte of information is read from, or written to, the FDC Data Register. Before each command byte is written, DIO (bit 6) must be low (indicating a data transfer from the processor) and RQM (bit 7) must be high (indicating that the FDC is ready for data). During the result phase, DIO must be high (indicating a data transfer to the processor) and RQM must also be high (indicating that data is ready for the processor).

NOTE:
After the 8272 receives a command byte, the RQM flag may remain set for 12 microseconds (with an 8 MHz clock). Software should not attempt to read the Main Status Register before this time interval has elapsed; otherwise, the software will erroneously assume that the FDC is ready to accept the next byte.

7.3 Sector Sizes

The 8272 does not support 128 byte sectors in the MFM (double-density) mode.

7.4 Write Clock

The FDC Write Clock input (WR CLK) must be present at all times.

7.5 Reset

The FDC Reset input (RST) must be held active during power-on reset while the RD and WR inputs are active. If the reset input becomes inactive while RD and WR are still active, the 8272 enters the test mode. Once activated, the test mode can only be deactivated by a power-down condition.

7.6 Drive Status

The 8272 constantly polls (starting after the power-on reset) all drives for changes in the drive ready status. At power-on, the FDC assumes that all drives are not ready. If a drive application requires that the ready line be strapped active, the FDC will generate an interrupt immediately after power is applied.

7.7 Gap Length

Only the gap 3 size is software programmable. All other gap sizes are fixed. In addition, different gap 3 sizes must be specified in format, read, write, and scan commands. Refer to Section 3 and Table 9 for gap size recommendations.

7.8 Seek Command

The drive busy flag in the Main Status Register remains set after a Seek command is issued until the Sense Interrupt Status command is issued (following reception of the seek complete interrupt).

The FDC does not perform implied seeks. Before issuing data read or write commands, the read/write head must be positioned over the correct cylinder. If the head is not positioned correctly, a cylinder address error is generated.

After issuing a step pulse, the 8272 resumes drive status polling. For correct stepper operation in this mode, the stepper motor must be constantly enabled. (Most drives provide a jumper to permit the stepper motor to be constantly enabled.)

7.9 Step Rate

The 8272 can emit a step pulse that is one millisecond faster than the rate programmed by the SRT parameter in the Specify command. This action may cause subsequent sector not found errors. The step rate time should be programmed to be 1 ms longer than the step rate time required by the drive.

7.10 Cable Length

A cable length of less than 10 feet is recommended for drive interfacing.

7.11 Scan Commands

The current 8272 has several problems when using the scan commands. These commands should not be used at this time.

7.12 Interrupts

When the processor receives an interrupt from the FDC, the FDC may be reporting one of two distinct events:

a) The beginning of the result phase of a previously requested read, write, or scan command.

b) An asynchronous event such as a seek/recalibrate completion, an attention, an abnormal command termination, or an invalid command.

These two cases are distinguished by the FDC busy flag (bit 4) in the Main Status Register. If the FDC busy flag is high, the interrupt is of type (a). If the FDC busy flag is low, the interrupt was caused by an asynchronous event (b).

A single interrupt from the FDC may signal more than one of the above events. After receiving an interrupt, the processor must continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are ferreted out and serviced.

7.13 Skip Flag (SK)

The skip flag is used during the execution of Read Data, Read Deleted Data, Read Track, and various Scan commands. This flag permits the FDC to skip unwanted sectors on a disk track.

When performing a Read Data, Read Track, or Scan command, a high SK flag indicates that the FDC is to skip over (not transfer) any sector containing a deleted data address mark. A low SK flag indicates that the FDC is to terminate the command (after reading all the data in the sector) when a deleted data address mark is encountered.

When performing a Read Deleted Data command, a high SK flag indicates that sectors containing normal data address marks are to be skipped. Note that his is just the opposite situation from that described in the last paragraph. When a data address mark is encountered during a Read Deleted Data command (and the SK flag is low), the FDC terminates the command after reading all the data in the sector.

7.14 Bad Track Maintenance

The 8272 does not internally maintain bad track information. The maintenance of this information must be performed by system software. As an example of typical bad track operation, assume that a media test determines that track 31 and track 66 of a given floppy disk are bad. When the disk is formatted for use, the system software formats physcial track 0 as logical cylinder 0 (C = 0 in the command phase parameters), physical track 1 as logical track 1 (C = 1), and so on, until physical track 30 is formatted as logical cylinder 30 (C = 30). Physical track 31 is bad and should be formatted as logical cylinder FF (indicating a bad track). Next, physical track 32 is formatted as logical cylinder 31, and so on, until physical track 67 is formatted as logical cylinder 64. Next, bad physical track 66 is formatted as logcial cylinder FF (another bad track marker), and physical track 67 is formatted as logical cylinder 65. This formatting continues until the last physical track (77) is formatted as logical cylinder 75. Normally, after this formatting is complete, the bad track information is stored in a prespecified area on the floppy disk (typically in a sector on track 0) so that the system will be able to recreate the bad track information when the disk is removed from the drive and reinserted at some later time.

To illustrate how the system software performs a transfer operation disk with bad tracks, assume that the disk drive head is positioned at track 0 and the disk described above is loaded into the drive. If a command to read track 36 is issued by an application program, the system software translates this read command into a seek to physical track 37 (since there is one bad track between 0 and 36, namely 31) followed by a read of logical cylinder 36. Thus, the cylinder parameter C is set to 37 for the Seek command and 36 for the Read Sector command.

7.15 Head Load versus Head Settle Times

The 8272 does not permit separate specification of the head load time and the head settle time. When the Specify command is issued for a given disk drive, the proper value for the HLT parameter is the maximum of the head load time and the head settle time.

APPENDIX A
SCHEMATICS

Power Distribution

Part	Ref Desig	+5	GND	+12	−12
8088	A2	40	1, 20		
8224	I6	9, 16	8		
8237-2	A6	31	20		
8251A	A9, B9, C9	26	4		
8253-5	A10	24	12		
8259A	B10	28	14		
8272	D10	40	20		
8284	A1	18	9		
8286	B6, F4	20	10		
2114	F1, F2, G1, G2, H1, H2, I1, I2	18	9		
2732A	D1, D2	24	12		
74LS00	E1	14	7		
74LS04	B2, E6, E8, F8	14	7		
74LS27	E2, E5	14	7		
74LS32	B1	14	7		
74LS74	A4, G5, H6	14	7		
74LS138	F3	16	8		
74LS139	E10	16	8		
74LS153	I3	16	8		
74LS157	F6	16	8		
74LS164	F5	14	7		
74LS173	G3	16	8		
74LS175	G4	16	8		
74LS240	G10	20	10		
74LS257	D3	16	8		
74LS367	C3, E9	16	8		
74LS373	B4, C4, D4, C6	20	10		
74LS393	I5, F7	14	7		
74S08	E4	14	7		
74S138	D6, E3	16	8		
7414	H7	14	7		
7438	H8, H9, H10	14	7		
1488	H3		7	14	1
1489	H4	14	7		
96LS02	G7	16	8		
96LS02	G6			16	8
LM358	H5			8	4

REFERENCES

1) Intel, "8272 Single/Double Density Floppy Disk Controller Data Sheet," Intel Corporation, 1980.

2) Intel, *iSBC 208 Hardware Reference Manual,* Manual Order No. 143078, Intel Corporation, 1980.

3) Intel, *iSBC 204 Flexible Diskette Controller Hardware Reference Manual,* Manual Order No. 9800568A, Intel Corporation, 1978.

4) Shugart, *SA800/801 Diskette Storage Drive OEM Manual,* Part No. 50574, Shugart Associates, 1977.

5) Shugart, *SA800/801 Diskette Storage Drive Theory of Operations,* Part No. 50664, Shugart Associates, 1977.

6) Shugart, *SA800 Series Diskette Storage Drive Double Density Design Guide,* Part No. 39000, Shugart Associates, 1977.

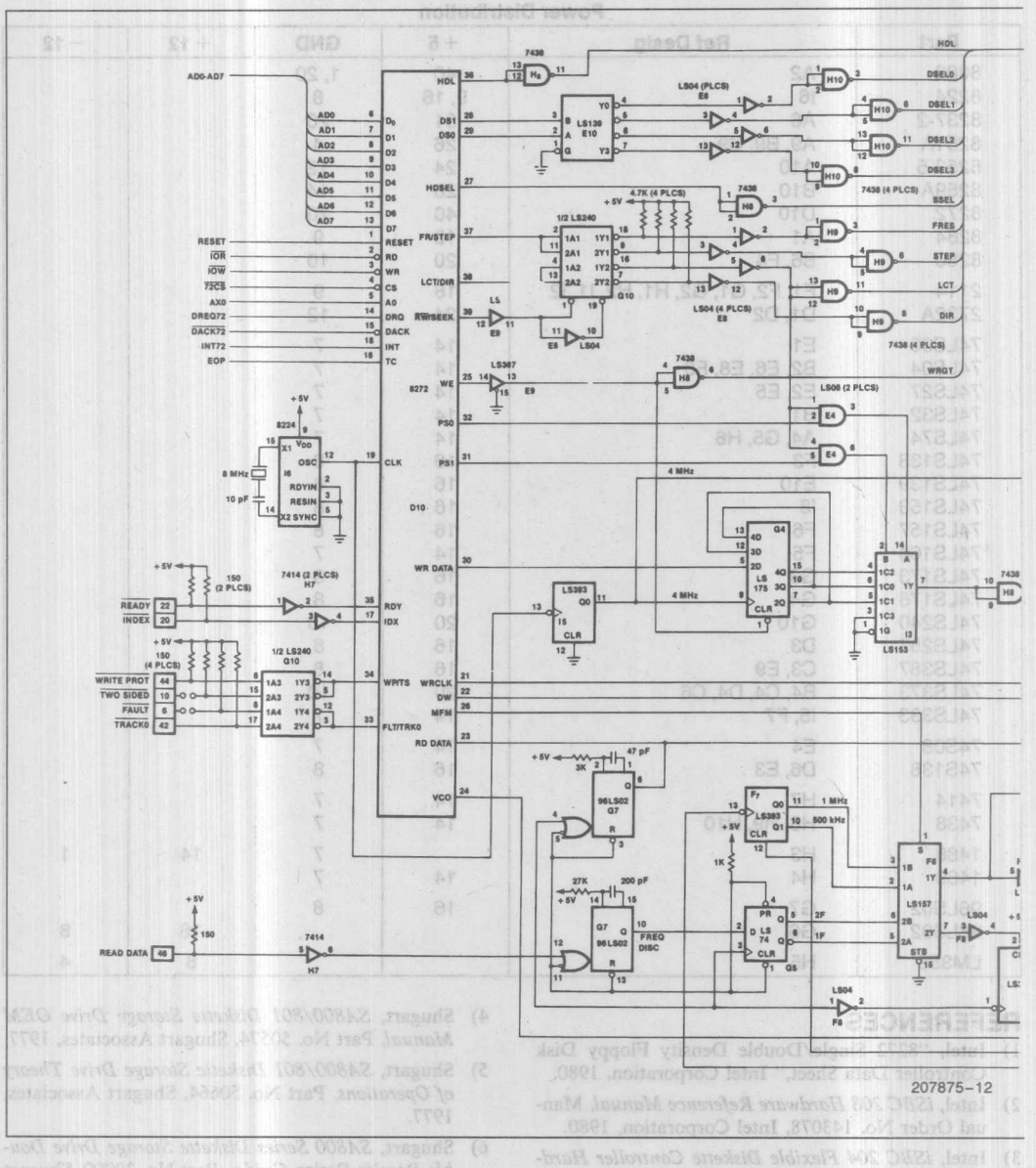
7) Shugart, "Application Notes for Shugart Dual VFO," Part No. 39101, Shugart Associates, 1980.

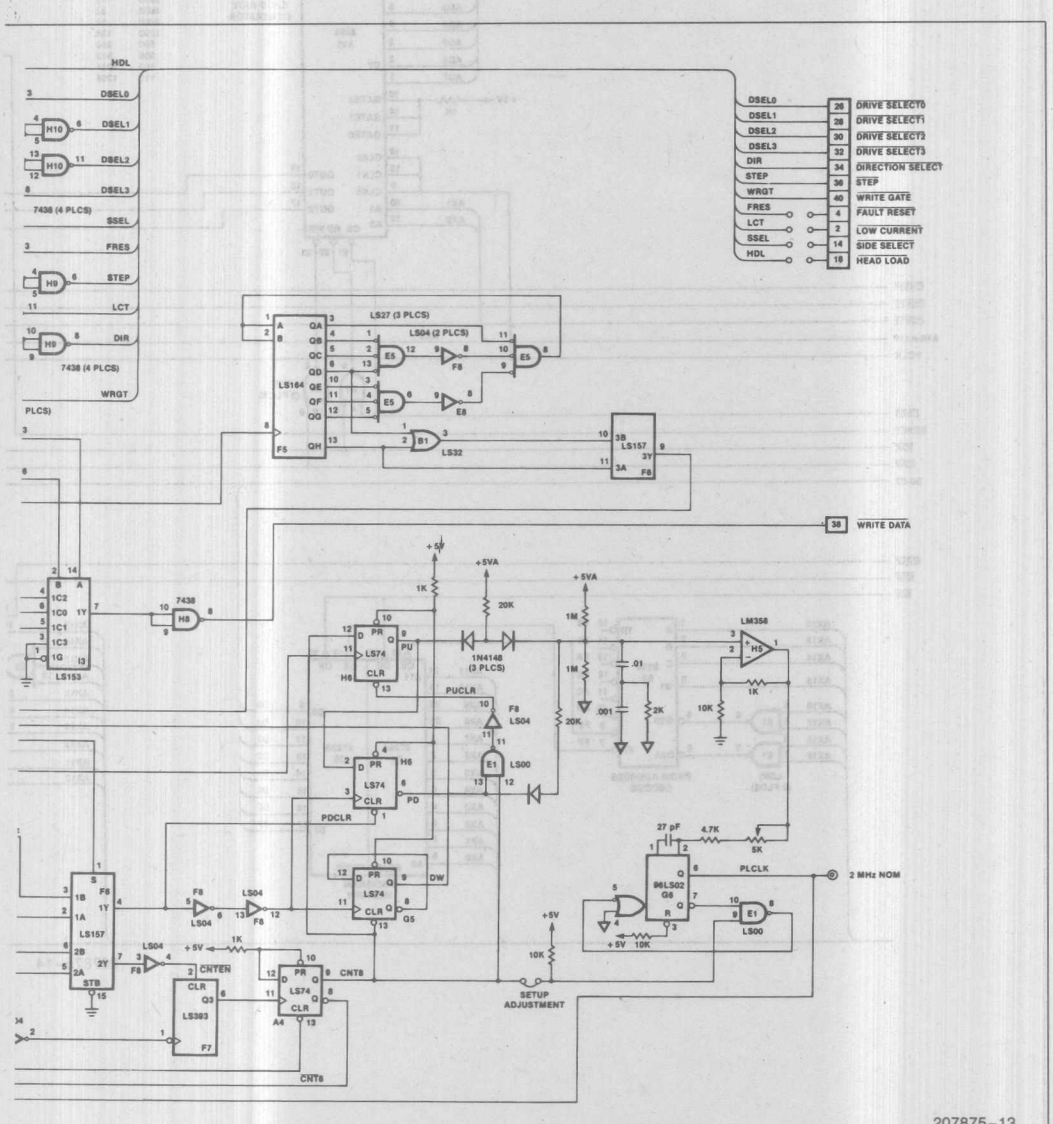
8) Pertec, "Soft-sector Formatting for PERTEC Flexible Disk Drives," Pertec Application Note, 1977.

9) Ausing Lesea and Rodnay Zaks, "Floppy-disc Controller Design Must Begin With the Basics," EDN, May 20, 1978.

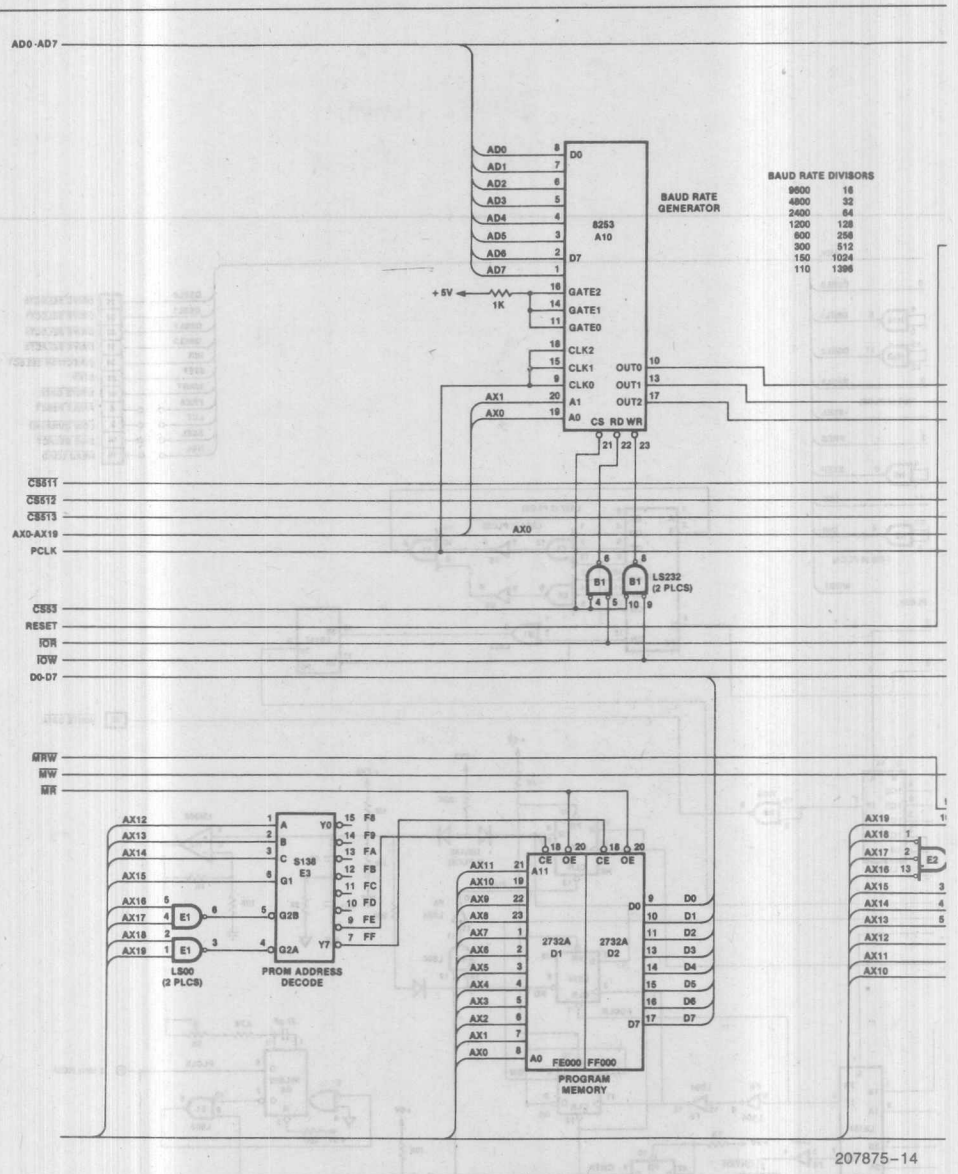
10) John Hoeppner and Larry Wall, "Encoding/Decoding Techniques Double Floppy Disc Capacity," Computer Design, Feb 1980.

11) John Zarrella, *System Architecture,* Microcomputer Applications, 1980.

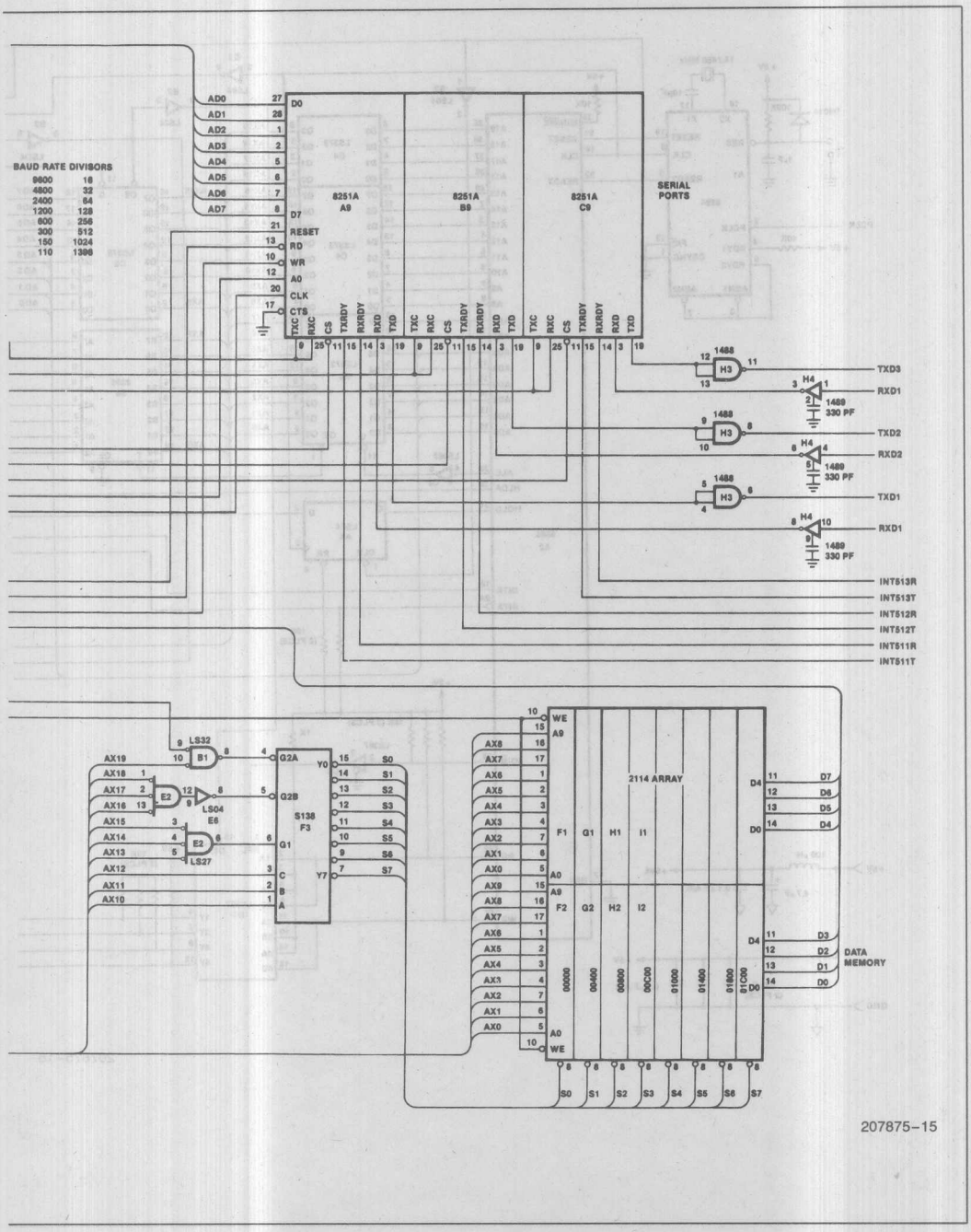



HDL
DSEL0
DSEL1
DSEL2
DSEL3
SSEL
FRES
STEP
LCT
DIR
WRGT
7438 (4 PLCS)
DRIVE SELECT0
DRIVE SELECT1
DRIVE SELECT2
DRIVE SELECT3
DIRECTION SELECT
STEP
WRITE GATE
FAULT RESET
LOW CURRENT
SIDE SELECT
HEAD LOAD
LS27 (3 PLCS)
LS04 (2 PLCS)
LS164
LS32
LS157
WRITE DATA
LS153
7438
+5V
+5VA
1K
20K
1M
1N4148
(3 PLCS)
LM358
.01
.001
2K
10K
LS74
PRCLR
PUCLR
PDCLR
LS04
LS00
27 pF
4.7K
5K
96LS02
PLCLK
2 MHz NOM
LS157
LS393
CNTEN
CNT8
SETUP
ADJUSTMENT

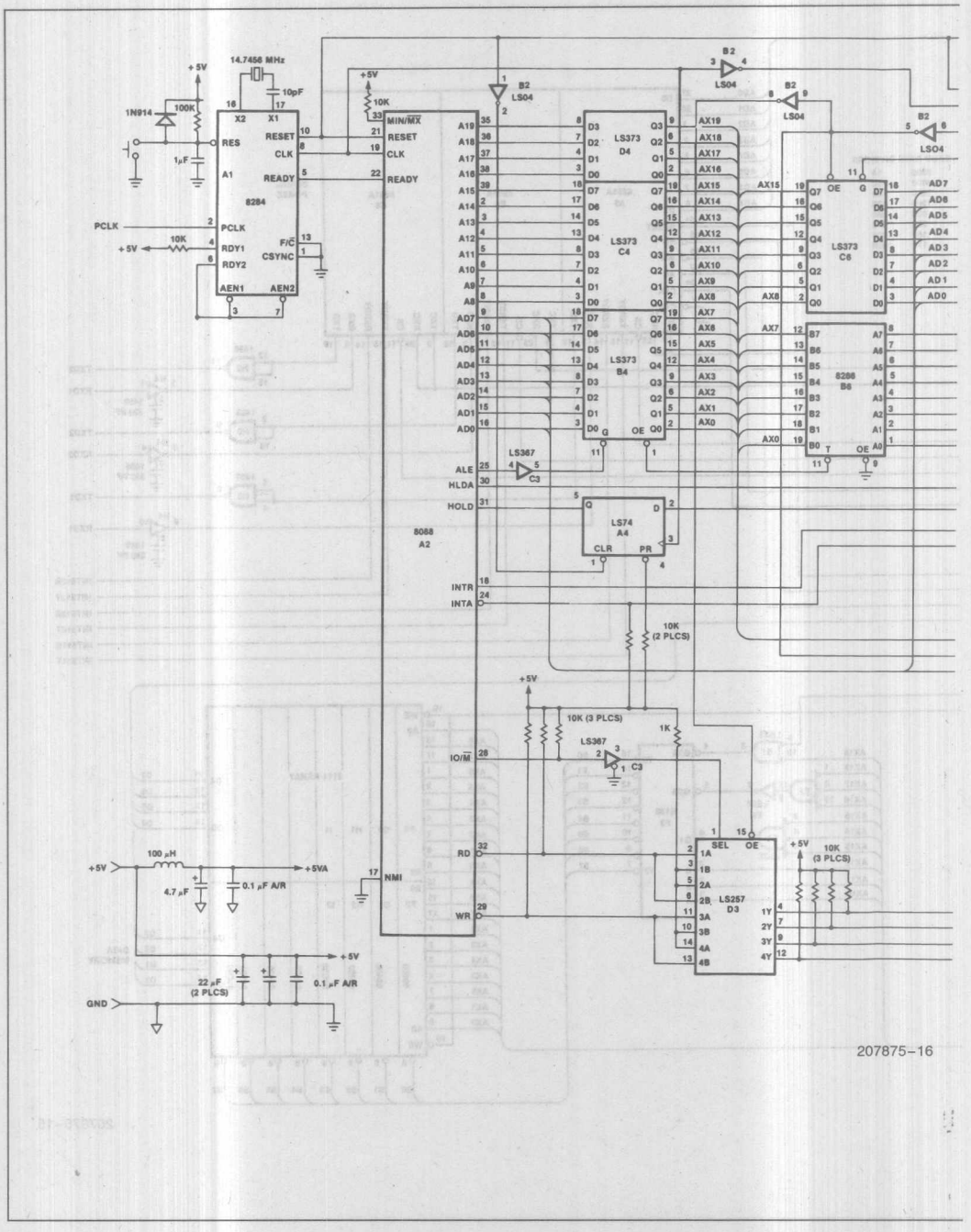
207875-13


AD0-AD7
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
D0
D7
8253
A10
BAUD RATE
GENERATOR
BAUD RATE DIVISORS
9600 16
4800 32
2400 64
1200 128
600 256
300 512
150 1024
110 1396
+5V
1K
GATE2
GATE1
GATE0
CLK2
CLK1
CLK0
A1
A0
OUT0
OUT1
OUT2
CS RD WR
AX1
AX0
CS511
CS512
CS513
AX0-AX19
PCLK
B1
LS232
(2 PLCS)
CS53
RESET
IOR
IOW
D0-D7
MRW
MW
MR
AX12
AX13
AX14
AX15
AX16
AX17
AX18
AX19
E1
LS00
(2 PLCS)
S138
E3
A
B
C
G1
G2B
G2A
Y0
Y7
F8
F9
FA
FB
FC
FD
FE
FF
PROM ADDRESS
DECODE
AX11
AX10
AX9
AX8
AX7
AX6
AX5
AX4
AX3
AX2
AX1
AX0
CE OE
A11
2732A
D1
2732A
D2
D0
D1
D2
D3
D4
D5
D6
D7
A0
FE000 FF000
PROGRAM
MEMORY
AX19
AX18
AX17
AX16
AX15
AX14
AX13
AX12
AX11
AX10
E2

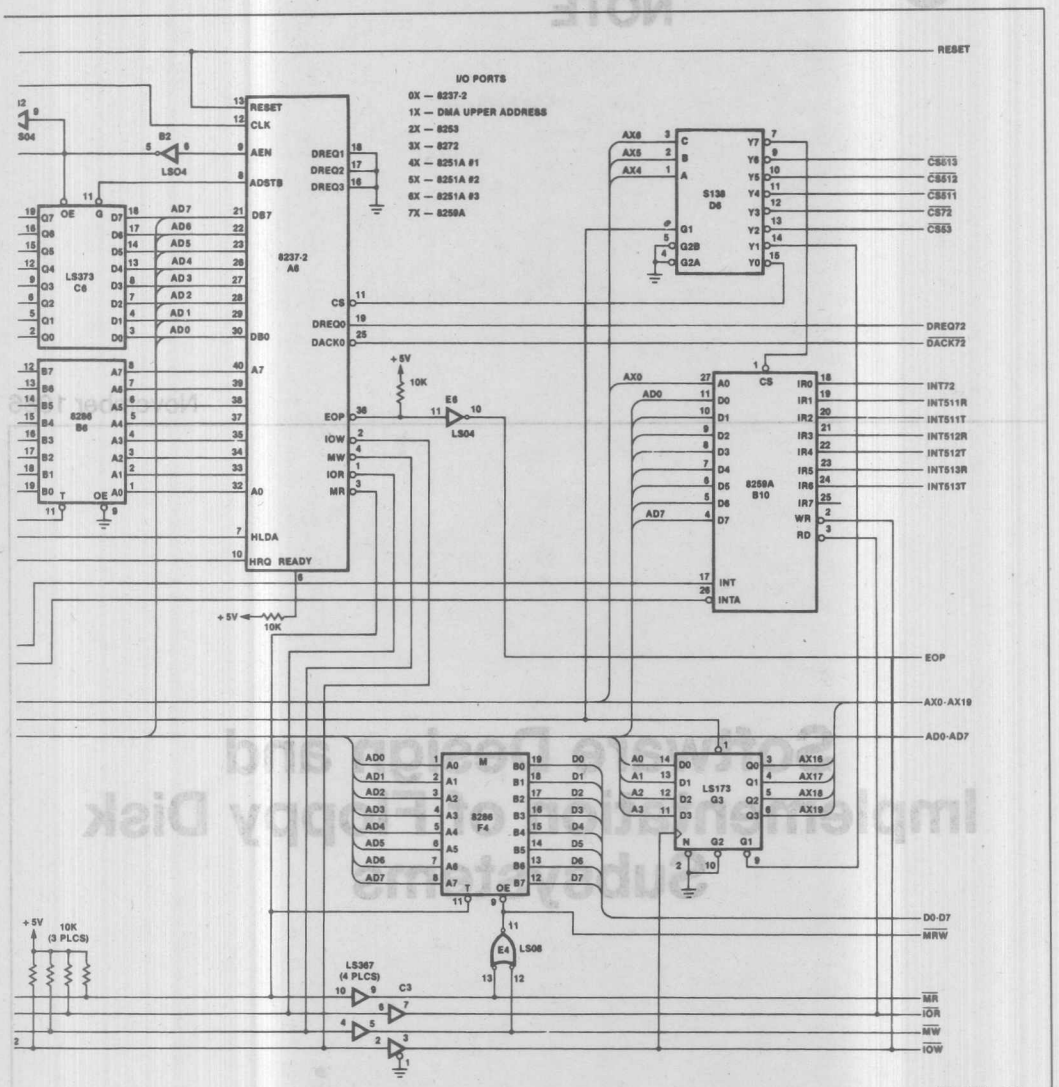
207875-14


BAUD RATE DIVISORS
9600 16
4800 32
2400 64
1200 128
600 256
300 512
150 1024
110 1396
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
D0
D7
RESET
RD
WR
A0
CLK
CTS
8251A A9
8251A B9
8251A C9
SERIAL PORTS
TXC
RXC
CS
TXRDY
RXRDY
RXD
TXD
1488
H3
H4
1489
330 PF
TXD3
RXD1
TXD2
RXD2
TXD1
RXD1
INT513R
INT513T
INT512R
INT512T
INT511R
INT511T
LS32
B1
E2
LS04
E6
LS27
S138
F3
G2A
G2B
G1
C
B
A
Y0
Y7
S0
S1
S2
S3
S4
S5
S6
S7
AX19
AX18
AX17
AX16
AX15
AX14
AX13
AX12
AX11
AX10
AX8
AX7
AX6
AX5
AX4
AX3
AX2
AX1
AX0
WE
A9
A0
F1
F2
Q1
Q2
H1
H2
I1
I2
2114 ARRAY
00000
00400
00800
00C00
01000
01400
01800
01C00
D7
D6
D5
D4
D3
D2
D1
D0
DATA MEMORY

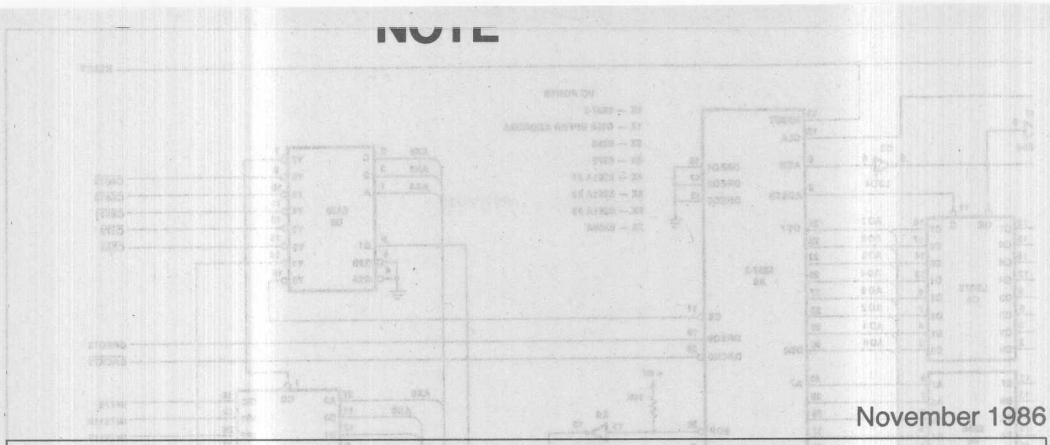
207875–15


14.7456 MHz
+5V
10pF
1N914
100K
1μF
X2
X1
RES
RESET
CLK
A1
READY
8284
PCLK
RDY1
RDY2
F/C
CSYNC
AEN1
AEN2
10K
MIN/MX
RESET
CLK
READY
8088
A2
A19
A18
A17
A16
A15
A14
A13
A12
A11
A10
A9
A8
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
ALE
HLDA
HOLD
INTR
INTA
LS373
D4
C4
B4
C6
B6
LS367
C3
LS74
A4
CLR
PR
LS04
B2
AX19
AX18
AX17
AX16
AX15
AX14
AX13
AX12
AX11
AX10
AX9
AX8
AX7
AX6
AX5
AX4
AX3
AX2
AX1
AX0
8286
10K (2 PLCS)
10K (3 PLCS)
1K
IO/M
RD
WR
NMI
LS257
D3
SEL
OE
100 μH
4.7 μF
0.1 μF A/R
+5VA
22 μF (2 PLCS)
GND

207875-16


RESET
I/O PORTS
0X — 8237-2
1X — DMA UPPER ADDRESS
2X — 8253
3X — 8272
4X — 8251A #1
5X — 8251A #2
6X — 8251A #3
7X — 8259A
8237-2 A8
LS373 C6
8286 B6
S138 D6
8259A B10
8286 F4
LS173 G3
LS367 (4 PLCS)
LS08 E4
LS04
10K
10K (3 PLCS)
DREQ72
DACK72
INT72
INT511R
INT511T
INT512R
INT512T
INT513R
INT513T
EOP
AX0-AX19
AD0-AD7
D0-D7
MRW
MR
IOR
MW
IOW

207875-17



November 1986

Software Design and Implementation of Floppy Disk Subsystems

Order Number: 207885-001

1.0 INTRODUCTION

Disk interface software is a major contributor to the efficient and reliable operation of a floppy disk subsystem. This software must be a well-designed compromise between the needs of the application software modules and the capabilities of the floppy disk controller (FDC). In an effort to meet these requirements, the implementation of disk interface software is often divided into several levels of abstraction. The purpose of this application note is to define these software interface levels and describe the design and implementation of a modular and flexible software driver for the 8272 FDC. This note is a companion to AP-116, "An Intelligent Data Base System Using the 8272".

The Physical Interface Level

The software interface level closest to the FDC hardware is referred to as the physical interface level. At this level, interface modules (often called disk drivers or disk handlers) communicate directly with the FDC device. Disk drivers accept floppy disk commands from other software modules, control and monitor the FDC execution of the commands, and finally return operational status information (at command termination) to the requesting modules.

In order to perform these functions, the drivers must support the bit/byte level FDC interface for status and data transfers. In addition, the drivers must field, classify, and service a variety of FDC interrupts.

The Logical Interface Level

System and application software modules often specify disk operation parameters that are not directly compatible with the FDC device. This software incompatibility is typically caused by one of the following:

1) The change from an existing FDC to a functionally equivalent design. Replacing a TTL based controller with an LSI device is an example of a change that may result in software incompatibilities.
2) The upgrade of an existing FDC subsystem to a higher capability design. An expansion from a single-sided, single-density system to a dual-sided, double-density system to increase data storage capacity is an example of such a system change.
3) The abstraction of the disk software interface to avoid redundancy. Many FDC parameters (in particular the density, gap size, number of sectors per track and number of bytes per sector) are fixed for a floppy disk (after formatting). In fact, in many systems these parameters are never changed during the life of the system.
4) The requirement to support a software interface that is independent of the type of disk attached to the system. In this case, a system generated ("logical") disk address (drive, head, cylinder, and sector numbers) must be mapped into a physical floppy disk address. For example, to switch between single- and dual-sided disks, it may be easier and more cost-effective for the software to treat the dual-sided disk as containing twice as many sectors per track (52) rather than as having two sides. With this technique, accesses to sectors 1 through 26 are mapped onto head 0 while accesses to sectors 27 through 52 are mapped onto head 1.
5) The necessity of supporting a bad track map. Since bad tracks depend on the disk media, the bad track mapping varies from disk to disk. In general, the system and application software should not be concerned with calculating bad track parameters. Instead, these software modules should refer to cylinders logically (0 through 76). The logical interface level procedures must map these cylinders into physical cylinder positions in order to avoid the bad tracks.

The key to logical interface software design is the mapping of the "logical disk interface" (as seen by the application software) into the "physical disk interface" (as implemented by the floppy disk drivers). This logical to physical mapping is tightly coupled to system software design and the mapping serves to isolate both applications and system software from the peculiarities of the FDC device. Typical logical interface procedures are described in Table 1.

The File System Interface Level

The file system typically comprises the highest level of disk interface software used by application programs. The file system is designed to treat the disk as a collection of named data areas (known as files). These files are cataloged in the disk directory. File system interface software permits the creation of new files and the deletion of existing files under software control. When a file is created, its name and disk address are entered into the directory; when a file is deleted, its name is removed from the directory. Application software requests the use of a file by executing an OPEN function. Once opened, a file is normally reserved for use by the requesting program or task and the file cannot be reopened by other tasks. When a task no longer needs to use an open file, the task closes the file, releasing it for use by other tasks.

Most file systems also support a set of file attributes that can be specified for each file. File attributes may be used to protect files (e.g., the WRITE PROTECT attri-

bute ensures that an existing file cannot accidentally be overwritten) and to supply system configuration information (e.g., a FORMAT attribute may specify that a file should automatically be created on a new disk when the disk is formatted).

At the file system interface level, application programs need not be explicitly aware of disk storage allocation techniques, block sizes, or file coding strategies. Only a "file name" must be presented in order to open, read or write, and subsequently close a file. Typical file system functions are listed in Table 2.

Table 1. Examples of Logical Interface Procedures

Name	Description
FORMAT DISK	Controls physical disk formatting for all tracks on a disk. Formatting adds FDC recognized cylinder, head, and sector addresses as well as address marks and data synchronization fields (gaps) to the floppy disk media.
RECALIBRATE	Moves the disk read/write head to track 0 (at the outside edge of the disk).
SEEK	Moves the disk read/write head to a specified logical cylinder. The logical and physical cylinder numbers may be different if bad track mapping is used.
READ STATUS	Indicates the status of the floppy disk drive and media. One important use of this procedure is to determine whether a floppy disk is dual-sided.
READ SECTOR	Reads one or more complete sectors starting at a specified disk address (drive, head, cylinder, and sector).
WRITE SECTOR	Writes one or more complete sectors starting at a specified disk address (drive, head, cylinder, and sector).

Table 2. Disk File System Functions

Name	Description
OPEN	Prepare a file for processing. If the file is to be opened for input and the file name is not found in the directory, an error is generated. If the file is opened for output and the file name is not found in the directory, the file is automatically created.
CLOSE	Terminate processing of an open file.
READ	Transfer Data from an open file to memory. The READ function is often designed to buffer one or more sectors of data from the disk drive and supply this data to the requesting program, as required.
WRITE	Transfer data from memory to an open file. The WRITE function is often designed to buffer data from the application program until enough data is available to fill a disk sector.
CREATE	Initialize a file and enter its name and attributes into the file directory.
DELETE	Remove a file from the directory and release its storage space.
RENAME	Change the name of a file in the directory.
ATTRIBUTE	Change the attributes of a file.
LOAD	Read a file of executable code into memory.
INITDISK	Initialize a disk by formatting the media and establishing the directory file, the bit map file, and other system files.

Scope of this Note

This application note directly addresses the logical and physical interface levels. A complete 8272 driver (including interrupt service software) is listed in Appendix A. In addition, examples of recalibrate, seek, format, read, and write logical interface level procedures are included as part of the exerciser program found in Appendix B. Wherever possible, specific hardware configuration dependencies are parametized to provide maximum flexibility without requiring major software changes.

2.0 Disk I/O Techniques

One of the most important software aspects of disk interfacing is the fixed sector size. (Sector sizes are fixed when the disk is formatted.) Individual bytes of disk storage cannot be read/written; instead, complete sectors must be transferred between the floppy disk and system memory.

Selection of the appropriate sector size involves a tradeoff between memory size, disk storage efficiency, and disk transfer efficiency. Basically, the following factors must be weighed:

1) Memory size. The larger the sector size, the larger the memory area that must be reserved for use during disk I/O transfers. For example, a 1K byte disk sector size requires that at least one 1K memory block be reserved for disk I/O.
2) Disk Storage efficiency. Both very large and very small sectors can waste disk storage space as follows. In disk file systems, space must be allocated somewhere on the disk to link the sectors of each file together. If most files are composed of many small sectors, a large amount of linkage overhead information is required. At the other extreme, when most files are smaller than a single disk sector, a large amount of space is wasted at the end of each sector.
3) Disk transfer efficiency. A file composed of a few large sectors can be transferred to/from memory more efficiently (faster and with less overhead) than a file composed of many small sectors.

Balancing these considerations requires knowledge of the intended system applications. Typically, for general purpose systems, sector sizes from 128 bytes to 1K bytes are used. For compatibility between single-density and double-density recording with the 8272 floppy disk controller, 256 byte sectors or 512 byte sectors are most useful.

FDC Data Transfer Interface

Three distinct software interface techniques may be used to interface system memory to the FDC device during sector data transfers:

1) DMA—In a DMA implementation, the software is only required to set up the DMA controller memory address and transfer count, and to initiate the data transfer. The DMA controller hardware handshakes with the processor/system bus in order to perform each data transfer.
2) Interrupt Driven—The FDC generates an interrupt when a data byte is ready to be transferred to memory, or when a data byte is needed from memory. It is the software's responsibility to perform appropriate memory reads/writes in order to transfer data from/to the FDC upon receipt of the interrupt.
3) Polling—Software responsibilities in the polling mode are identical to the responsibilities in the interrupt driven mode. The polling mode, however, is used when interrupt service overhead (context switching) is too large to support the disk data rate. In this mode, the software determines when to transfer data by continually polling a data request status flag in the FDC status register.

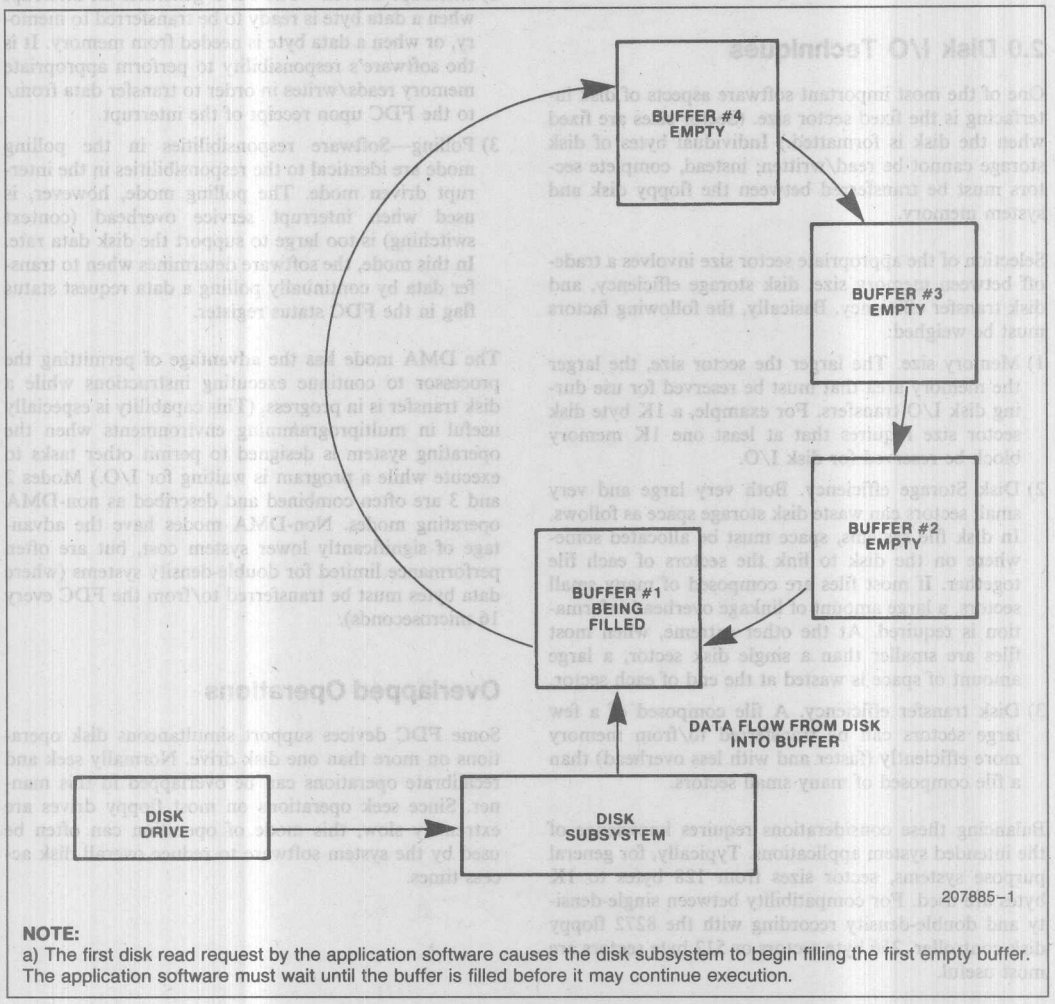
The DMA mode has the advantage of permitting the processor to continue executing instructions while a disk transfer is in progress. (This capability is especially useful in multiprogramming environments when the operating system is designed to permit other tasks to execute while a program is waiting for I/O.) Modes 2 and 3 are often combined and described as non-DMA operating modes. Non-DMA modes have the advantage of significantly lower system cost, but are often performance limited for double-density systems (where data bytes must be transferred to/from the FDC every 16 microseconds).

Overlapped Operations

Some FDC devices support simultaneous disk operations on more than one disk drive. Normally seek and recalibrate operations can be overlapped in this manner. Since seek operations on most floppy drives are extremely slow, this mode of operation can often be used by the system software to reduce overall disk access times.

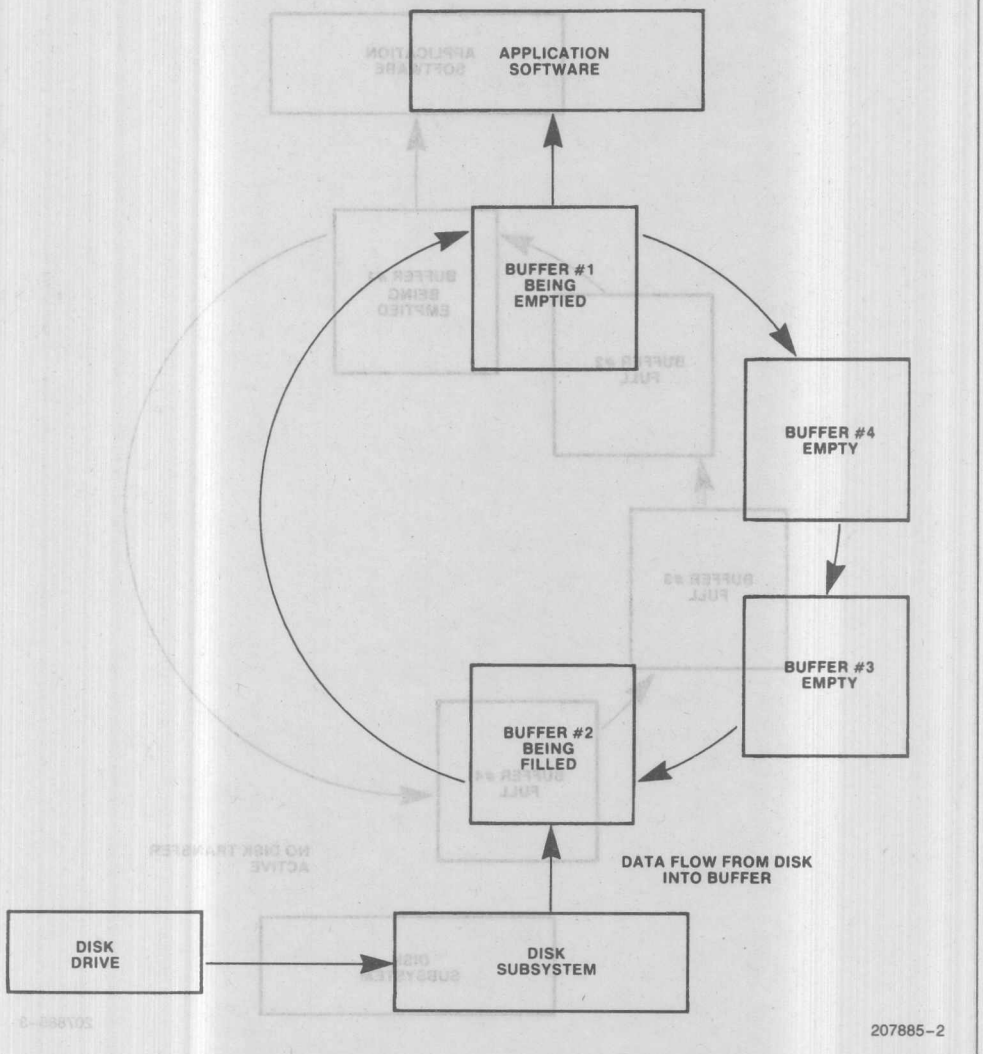
Buffers

The buffer concept is an extremely important element in advanced disk I/O strategies. A buffer is nothing more than a memory area containing the same amount of data as a disk sector contains. Generally, when an application program requests data from a disk, the system software allocates a buffer (memory area) and transfers the data from the appropriate disk sector into the buffer. The address of the buffer is then returned to the application software. In the same manner, after the application program has filled a buffer for output, the buffer address is passed to the system software, which writes data from the buffer into a disk sector. In multitasking systems, multiple buffers may be allocated from a buffer pool. In these systems, the disk controller is often requested to read ahead and fill additional data buffers while the application software is processing a previous buffer. Using this technique, system software attempts to fill buffers before they are needed by the application programs, thereby eliminating program waits during I/O transfers. Figure 1 illustrates the use of multiple buffers in a ring configuration.



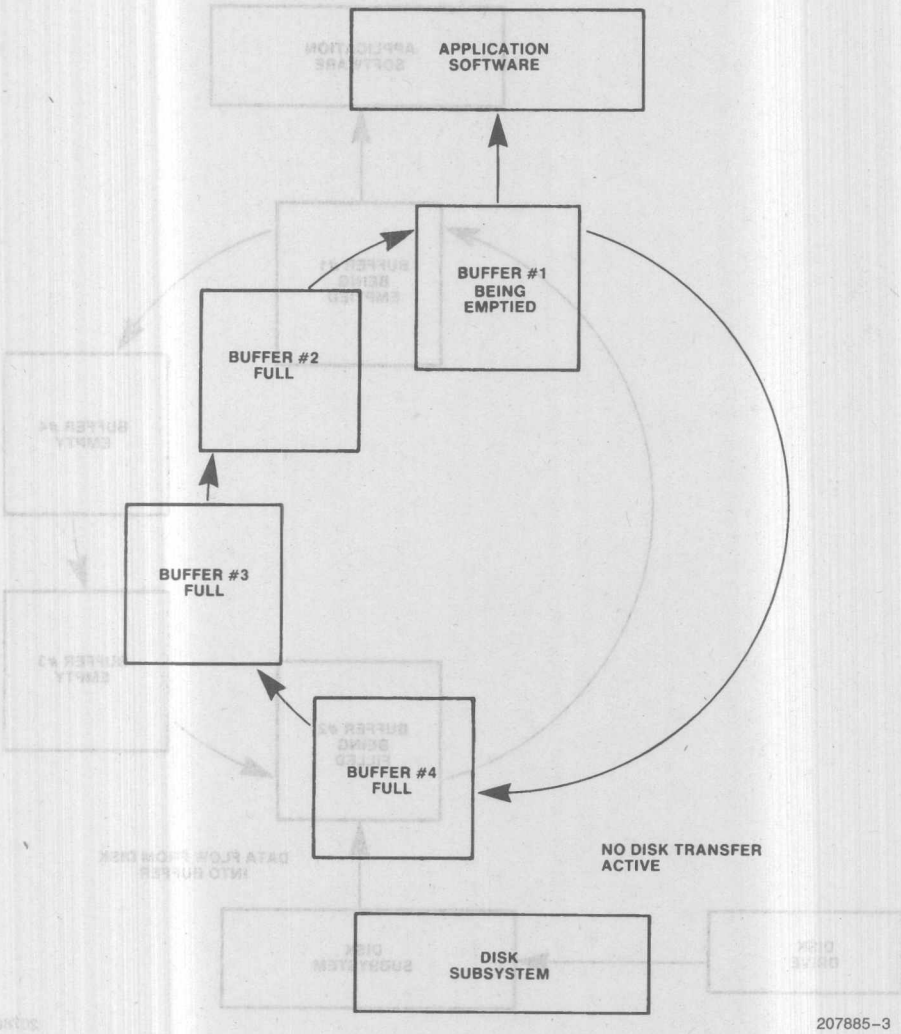
NOTE:
a) The first disk read request by the application software causes the disk subsystem to begin filling the first empty buffer. The application software must wait until the buffer is filled before it may continue execution.

Figure 1. Using Multiple Memory Buffers for Disk I/O



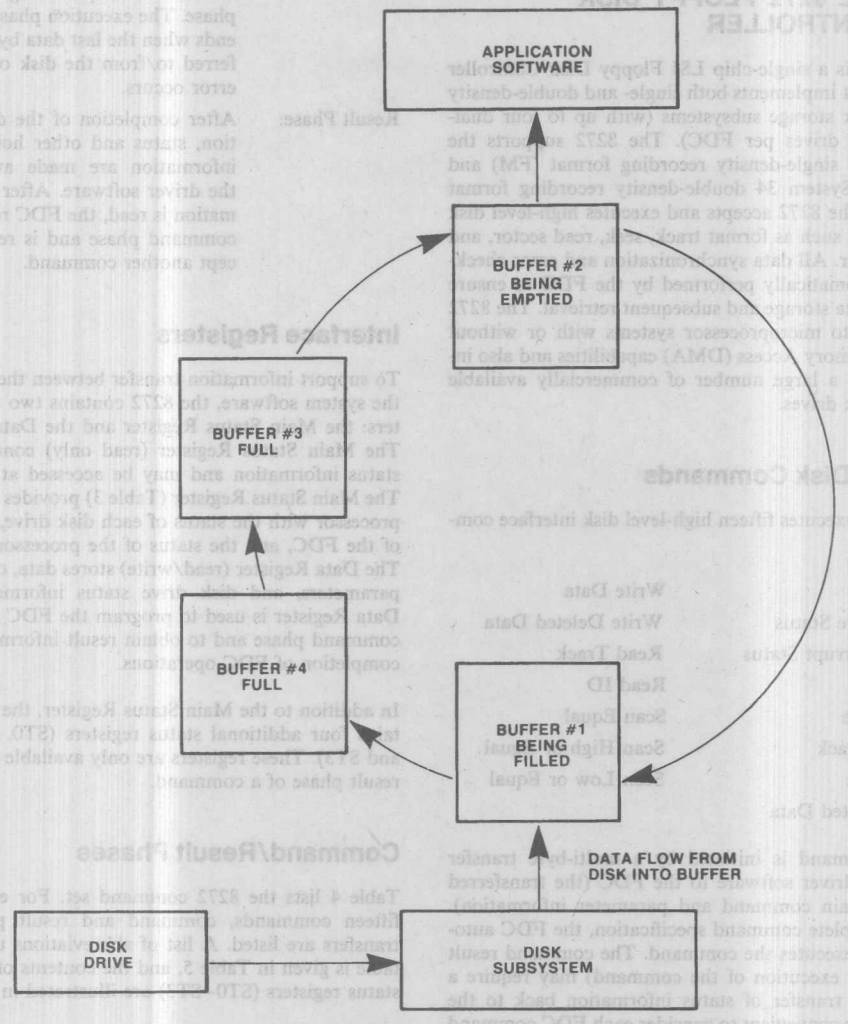
NOTE:
b) After the first buffer is filled, the disk system continues to transfer disk data into the next buffer while the application software begins operating on the first full buffer.

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)



NOTE:
c) When all empty buffers have been filled, disk activity is stopped until the application software releases one or more buffers for reuse.

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)



NOTE:
d) When the application software releases a buffer (for reuse), the disk subsystem begins a disk sector read to refill the buffer. This strategy attempts to anticipate application software needs by maintaining a sufficient number of full data buffers in order to minimize data transfer delays. If disk data is already in memory when the application software requests it, no disk transfer delays are incurred.

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)

3.0 THE 8272 FLOPPY DISK CONTROLLER

The 8272 is a single-chip LSI Floppy Disk Controller (FDC) that implements both single- and double-density floppy disk storage subsystems (with up to four dual-sided disk drives per FDC). The 8272 supports the IBM 3740 single-density recording format (FM) and the IBM System 34 double-density recording format (MFM). The 8272 accepts and executes high-level disk commands such as format track, seek, read sector, and write sector. All data synchronization and error checking is automatically performed by the FDC to ensure reliable data storage and subsequent retrieval. The 8272 interfaces to microprocessor systems with or without Direct Memory Access (DMA) capabilities and also interfaces to a large number of commercially available floppy disk drives.

Floppy Disk Commands

The 8272 executes fifteen high-level disk interface commands:

Specify	Write Data
Sense Drive Status	Write Deleted Data
Sense Interrupt Status	Read Track
Seek	Read ID
Recalibrate	Scan Equal
Format Track	Scan High or Equal
Read Data	Scan Low or Equal
Read Deleted Data	

Each command is initiated by a multi-byte transfer from the driver software to the FDC (the transferred bytes contain command and parameter information). After complete command specification, the FDC automatically executes the command. The command result data (after execution of the command) may require a multi-byte transfer of status information back to the driver. It is convenient to consider each FDC command as consisting of the following three phases:

Command Phase: The driver transfers to the FDC all the information required to perform a particular disk operation. The 8272 automatically enters the command phase after RESET and following the completion of the result phase (if any) of a previous command.

Execution Phase: The FDC performs the operation as instructed. The execution phase is entered immediately after the last command parameter is written to the FDC in the preceding command phase. The execution phase normally ends when the last data byte is transferred to/from the disk or when an error occurs.

Result Phase: After completion of the disk operation, status and other housekeeping information are made available to the driver software. After this information is read, the FDC reenters the command phase and is ready to accept another command.

Interface Registers

To support information transfer between the FDC and the system software, the 8272 contains two 8-bit registers: the Main Status Register and the Data Register. The Main Status Register (read only) contains FCD status information and may be accessed at any time. The Main Status Register (Table 3) provides the system processor with the status of each disk drive, the status of the FDC, and the status of the processor interface. The Data Register (read/write) stores data, commands, parameters, and disk drive status information. The Data Register is used to program the FDC during the command phase and to obtain result information after completion of FDC operations.

In addition to the Main Status Register, the FDC contains four additional status registers (ST0, ST1, ST2, and ST3). These registers are only available during the result phase of a command.

Command/Result Phases

Table 4 lists the 8272 command set. For each of the fifteen commands, command and result phase data transfers are listed. A list of abbreviations used in the table is given in Table 5, and the contents of the result status registers (ST0–ST3) are illustrated in Table 6.

The bytes of data which are sent to the 8272 by the drivers during the command phase, and are read out of the 8272 in the result phase, must occur in the order shown in Table 4. That is, the command code must be sent first and the other bytes sent in the prescribed sequence. All bytes of the command and result phases must be read/written as described. After the last byte of data in the command phase is sent to the 8272 the execution phase automatically starts. In a similar fashion, when the last byte of data is read from the 8272 in the result phase, the result phase is automatically ended and the 8272 reenters the command phase.

It is important to note that during the result phase all bytes shown in Table 4 must be read. The Read Data

command, for example, has seven bytes of data in the result phase. All seven bytes must be read in order to successfully complete the Read Data command. The 8272 will not accept a new command until all seven bytes have been read. The number of command and result bytes varies from command-to-command.

In order to read data from, or write data to, the Data Register during the command and result phases, the software driver must examine the Main Status Register to determine if the Data Register is available. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command word may be written into the 8272. Many of the commands require multiple bytes, and as a result, the Main Status Register must be read prior to each byte transfer to the 8272. To read status bytes during the result phase, DIO and RQM in the Main Status Register must both be high. Note, checking the Main Status Register in this manner before each byte transfer to/from the 8272 is required only in the command and result phases, and is NOT required during the execution phase.

Table 3. Main Status Register Bit Definitions

Bit Number	Symbol	Description
0	D_0B	Disk Drive 0 Busy. Disk Drive 0 is seeking.
1	D_1B	Disk Drive 1 Busy. Disk Drive 1 is seeking.
2	D_2B	Disk Drive 2 Busy. Disk Drive 2 is seeking.
3	D_3B	Disk Drive 3 Busy. Disk Drive 3 is seeking.
4	CB	FDC Busy. A read or write command is in progress.
5	NDM	Non-DMA Mode. The FDC is in the non-DMA mode when this flag is set (1). This flag is set only during the execution phase of commands in the non-DMA mode. Transition of this flag to a zero (0) indicates that the execution phase has ended.
6	DIO	Data Input/Output. Indicates the direction of a data transfer between the FDC and the Data Register. When DIO is set (1), data is read from the Data Register by the processor; when DIO is reset (0), data is written from the processor to the Data Register.
7	RQM	Request for Master. When set (1), this flag indicates that the Data Register is ready to send data to, or receive data from, the processor.

Table 4. 8272 Command Set

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after command execution
	R				H					
	R				R					
	R				N					
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EC 1					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the main-system and the FDD
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system. FDC reads the complete track contents from the physical index mark to EOT
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the track is stored in Data Register
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information during Execution Phase
	R				H					
	R				R					
	R				N					
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				N					Bytes/Sector
	W				SC					Sectors/Track
	W				GPL					Gap 3
	W				D					Filter Byte
Execution										FDC formats an entire track
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					In this case, the ID information has no meaning
	R				H					
	R				R					
	R				N					
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data compared between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

207885-34

NOTE:
1. A_0 = 1 for all operations.

Table 4. 8272 Command Set (Continued)

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
SCAN LOW OR EQUAL										
Command	W	MT	MFM	SK	1	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data compared between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
SCAN HIGH OR EQUAL										
Command	W	MT	MFM	SK	1	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data compared between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head retracted to Track 0
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R				ST 0					Status information at the end of each seek operation about the FDC
	R				C					
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	SPT ← HUT								Timer Settings
	W	HLT → ND								
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Result	R				ST 3					Status information about the FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					
Execution										Head is positioned over proper Cylinder on Diskette
INVALID										
Command	W				Invalid Codes					Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R				ST 0					ST 0 = 80 (16)

207885-35

NOTE:
1. A_0 = 1 for all operations.

Table 5. Command/Result Parameter Abbreviations

Symbol	Description
C	**CYLINDER ADDRESS.** The currently selected cylinder address (0 to 76) on the disk.
D	**DATA PATTERN.** The pattern to be written in each sector data field during formatting.
DS0, DS1	**DISK DRIVE SELECT.** **DS1 DS0** 0 0 Drive 0 0 1 Drive 1 1 0 Drive 2 1 1 Drive 3
DTL	**SPECIAL SECTOR SIZE.** During the execution of disk read/write commands, this parameter is used to temporarily alter the effective disk sector size. By setting N to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the disk) is larger than DTL specifies, the remainder of the actual sector is not passed to the system during read commands; during write commands, the remainder of the actual sector is written with all-zeroes bytes. DTL should be set to FF hexadecimal when N is not zero.
EOT	**END OF TRACK.** The final sector number of the current track.
GPL	**GAP LENGTH.** The gap 3 size. (Gap 3 is the space between sectors).
H	**HEAD ADDRESS.** Selected head: 0 or 1 (disk side 0 or 1, respectively) as encoded in the sector ID field.
HLT	**HEAD LOAD TIME.** Defines the time interval that the FDC waits after loading the head before initiating the read or write operation. Programmable from 2 to 254 milliseconds (in increments of 2 ms).
HUT	**HEAD UNLOAD TIME.** Defines the time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Programmable from 16 to 240 milliseconds (in increments of 16 ms).
MFM	**MFM/FM MODE SELECTOR.** Selects MFM double-density recording mode when high, FM single-density mode when low.
MT	**MULTI-TRACK SELECTOR.** When set, this flag selects the multi-track operating mode. In this mode (used only with dual-sided disks), the FDC treats a complete cylinder (under both read/write head 0 and read/write head 1) as a single track. The FDC operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set (high), a multi-sector read operation will automatically continue to the first sector under head 1 when the FDC finishes operating on the last sector under head 0.
N	**SECTOR SIZE CODE.** The number of data bytes within a sector.
ND	**NON-DMA MODE FLAG.** When set (1), this flag inidcates that the FDC is to operate in the non-DMA mode. In this mode, the processor participates in each data transfer (by means of an interrupt or by polling the RQM flag in the Main Status Register). When reset (0), the FDC interfaces to a DMA controller.
R	**SECTOR ADDRESS.** Specifies the sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of the first sector to be read or written.
SC	**NUMBER OF SECTORS PER TRACK.** Specifies the number of sectors per track to be initialized by the Format Track command.
SK	**SKIP FLAG.** When this flag is set, sectors containing deleted data address marks will automatically be skipped during the execution of multi-sector Read Data or Scan commands. In the same manner, a sector containing a data address mark will automatically be skipped during the execution of a multi-sector Read Deleted Data command.
SRT	**STEP RATE INTERVAL.** Defines the time interval between step pulses issued by the FDC (track-to-track access time). Programmable from 1 to 16 milliseconds (in increments of 1 ms).
ST0, ST1, ST2, ST3	**STATUS REGISTER 0–3.** Registers within the FDC that store status information after a command has been executed. This status information is available to the processor during the Result Phase after command execution. These registers may only be read after a command has been executed (in the exact order shown in Table 4 for each command). These registers should not be confused with the Main Status Register.
STP	**SCAN SECTOR INCREMENT.** During Scan operations, this parameter is added to the current sector number in order to determine the next sector to be scanned.

Table 6. Status Register Definitions

Bit Number	Symbol	Description
STATUS REGISTER 0		
7, 6	IC	**INTERRUPT CODE.** 00— Normal termination of command. The specified command was properly executed and completed without error. 01— Abnormal termination of command. Command execution was started but could not be successfully completed. 10— Invalid command. The requested command could not be executed. 11— Abnormal termination. During command execution, the disk drive ready signal changed state.
5	SE	**SEEK END.** This flag is set (1) when the FDC has completed the Seek command and the read/write head is positioned over the correct cylinder.
4	EC	**EQUIPMENT CHECK ERROR.** This flag is set (1) if a fault signal is received from the disk drive or if the track 0 signal is not received from the disk drive after 77 step pulses (Recalibrate command).
3	NR	**NOT READY ERROR.** This flag is set if a read or write command is issued and either the drive is not ready or the command specifies side 1 (head 1) of a single-sided disk.
2	H	**HEAD ADDRESS.** The head address at the time of the interrupt.
1, 0	DS1, DS0	**DRIVE SELECT.** The number of the drive selected at the time of the interrupt.
STATUS REGISTER 1		
7	EN	**END OF TRACK ERROR.** This flag is set if the FDC attempts to access a sector beyond the final sector of the track.
6		**UNDEFINED**
5	DE	**DATA ERROR.** Set when the FDC detects a CRC error in either the ID field or the data field of a sector.
4	OR	**OVERRUN ERROR.** Set (during data transfers) if the FDC does not receive DMA or processor service within the specified time interval.
3		**UNDEFINED**
2	ND	**SECTOR NOT FOUND ERROR.** This flag is set by any of the following conditions. a) The FDC cannot locate the sector specified in the Read Data, Read Deleted Data, or Scan command. b) The FDC cannot locate the starting sector specified in the Read Track command. c) The FDC cannot read the ID field without error during a Read ID command.
1	NW	**WRITE PROTECT ERROR.** This flag is set if the FDC detects a write protect signal from the disk drive during the execution of a Write Data, Write Deleted Data, or Format Track command.
0	MA	**MISSING ADDRESS MARK ERROR.** This flag is set by either of the following conditions: a) The FDC cannot detect the ID address mark on the specified track (after two rotations of the disk). b) The FDC cannot detect the data address mark or deleted data address mark on the specified track. (See also the MD bit of Status Register 2.)

Table 6. Status Register Definitions (Continued)

Bit Number	Symbol	Description
STATUS REGISTER 2		
7		**UNDEFINED**
6	CM	**CONTROL MARK.** This flag is set when the FDC encounters one of the following conditions: a) A deleted data address mark during the execution of a Read Data or Scan command. b) A data address mark during the execution of a Read Deleted Data command.
5	DD	**DATA ERROR.** Set (1) when the FDC detects a CRC error in a sector data field. This flag is not set when a CRC error is detected in the ID field.
4	WC	**CYLINDER ADDRESS ERROR.** Set when the cylinder address from the disk sector ID field is different from the current cylinder address maintained within the FDC.
3	SH	**SCAN HIT.** Set during the execution of the Scan command if the scan condition is satisfied.
2	SN	**SCAN NOT SATISFIED.** Set during execution of the Scan command if the FDC cannot locate a sector on the specified cylinder that satisfies the scan condition.
1	BC	**BAD TRACK ERROR.** Set when the cylinder address from the disk sector ID field is FF hexadecimal and this cylinder address is different from the current cylinder address maintained within the FDC. This all "ones" cylinder number indicates a bad track (one containing hard errors) according to the IBM soft-sectored format specifications.
0	MD	**MISSING DATA ADDRESS MARK ERROR.** Set if the FDC cannot detect a data address mark or deleted data address mark on the specified track.
STATUS REGISTER 3		
7	FT	**FAULT.** This flag indicates the status of the fault signal from the selected disk drive.
6	WP	**WRITE PROTECTED.** This flag indicates the status of the write protect signal from the selected disk drive.
5	RDY	**READY.** This flag indicates the status of the ready signal from the selected disk drive.
4	T0	**TRACK 0.** This flag indicates the status of the track 0 signal from the selected disk drive.
3	TS	**TWO-SIDED.** This flag indicates the status of the two-sided signal from the selected disk drive.
2	H	**HEAD ADDRESS.** This flag indicates the status of the side select signal for the currently selected disk drive.
1, 0	DS1, DS0	**DRIVE SELECT.** Indicates the currently selected disk drive number.

Execution Phase

All data transfers to (or from) the floppy drive occur during the execution phase. The 8272 has two primary modes of operation for data transfers (selected by the specify command):

1) DMA mode

2) non-DMA mode

In the DMA mode, execution phase data transfers are handled by the DMA controller hardware (invisible to the driver software). The driver software, however, must set all appropriate DMA controller registers prior to the beginning of the disk operation. An interrupt is generated by the 8272 after the last data transfer, indicating the completion of the execution phase, and the beginning of the result phase.

In the non-DMA mode, transfer requests are indicated by generation of an interrupt and by activation of the RQM flag (bit 7 in the Main Status Register). The interrupt signal can be used for interrupt-driven systems and RQM can be used for polled systems. The driver software must respond to the transfer request by reading data from, or writing data to, the FDC. After completing the last transfer, the 8272 generates an interrupt to indicate the beginning of the result phase. In the non-DMA mode, the processor must activate the "terminal count" (TC) signal to the FDC (normally by means of an I/O port) after the transfer request for the last data byte has been received (by the driver) and before the appropriate data byte has been read from (or written to) the FDC.

In either mode of operation (DMA or non-DMA), the execution phase ends when a "terminal count" signal is sensed by the FDC, when the last sector on a track (the EOT parameter—Table 4) has been read or written, or when an error occurs.

Multi-Sector and Multi-Track Transfers

During disk read/write transfers (Read Data, Write Data, Read Deleted Data, and Write Deleted Data), the FDC will continue to transfer data from sequential sectors until the TC input is sensed. In the DMA mode, the TC input is normally set by the DMA controller. In the non-DMA mode, the processor directly controls the FDC TC input as previously described. Once the TC input is received, the FDC stops requesting data transfers (from the system software or DMA controller). The FDC, however, continues to read data from, or write data to, the floppy disk until the end of the current disk sector. During a disk read operation, the data read from the disk (after reception of the TC input) is discarded, but the data CRC is checked for errors; during a disk write operation, the remainder of the sector is filled with all-zero bytes.

If the TC signal is not received before the last byte of the current sector has been transferred to/from the system, the FDC increments the sector number by one and initiates a read or write command for this new disk sector.

The FDC is also designed to operate in a multi-track mode for dual-sided disks. In the multi-track mode (specified by means of the MT flag in the command byte—Table 4) the FDC will automatically increment the head address (from 0 to 1) when the last sector (on the track under head 0) has been read or written. Reading or writing is then continued on the first sector (sector 1) of head 1.

Drive Status Polling

After the power-on reset, the 8272 automatically enters a drive status polling mode. If a change in drive status is detected (all drives are assumed to be "not ready" at power-on), an interrupt is generated. The 8272 continues this status polling between command executions (and between step pulses in the Seek command). In this manner, the 8272 automatically notifies the system software whenever a floppy disk is inserted, removed, or changed by the operator.

Command Details

During the command phase, the Main Status Register must be polled by the driver software before each byte is written into the Data Register. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be

low and high, respectively, before each byte of the command may be written into the 8272. The beginning of the execution phase for any of these commands will cause DIO to be set high and RQM to be set low.

Operation of the FDC commands is described in detail in Application Note AP-116, "An Intelligent Data Base System Using the 8272".

Invalid Commands

If an invalid (undefined) command is sent to the FDC, the FDC will terminate the command. No interrupt is generated by the 8272 during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both set indicating to the processor that the 8272 is in the result phase and the contents of Status Register 0 must be read. When the processor reads Status Register 0 it will find an 80H code indicating that an invalid command was received. The driver software in Appendix B checks each requested command and will not issue an invalid command to the 8272.

A Sense Interrupt Status command must be sent after a Seek or Recalibrate interrupt; otherwise the FDC will consider the next command to be an invalid command. Also, when the last "hidden" interrupt has been serviced, further Sense Interrupt Status commands will result in invalid command codes.

4.0 8272 PHYSICAL INTERFACE SOFTWARE

PL/M software driver listings for the 8272 FDC are contained in Appendix A. These drivers have been designed to operate in a DMA environment (as described in Application Note AP-116, "An Intelligent Data Base System Using the 8272"). In the following paragraphs, each driver procedure is described. (A description of the driver data base variables is given in Table 7.) In addition, the modifications necessary to reconfigure the drivers for operation in a polled environment are discussed.

INITIALIZE$DRIVERS

This initialization procedure must be called before any FDC operations are attempted. This module initializes the DRIVE$READY, DRIVE$STATUS$CHANGE, OPERATION$IN$PROGRESS, and OPERATION$COMPLETE arrays as well as the GLOBAL$DRIVE$NO variable.

EXECUTE$DOCB

This procedure contains the main 8272 driver control software and handles the execution of a complete FDC command. EXECUTE$DOCB is called with two parameters: a) a pointer to a disk operation control block and b) a pointer to a result status byte. The format of the disk operation control block is illustrated in Figure 2 and the result status codes are described in Table 8.

Before starting the command phase for the specified disk operation, the command is checked for validity and to determine whether the FDC is busy. (For an overlapped operation, if the FDC BUSY flag is set—in the Main Status Register—the command cannot be started; non-overlapped operations cannot be started if the FDC BUSY flag is set, if any drive is in the process of seeking/recalibrating, or if an operation is currently in progress on the specified drive.)

After these checks are made, interrupts are disabled in order to set the OPERATIONINPROGRESS flag, reset the OPERATION$COMPLETE flag, load a pointer to the current operation control block into the OPERATION$DOCB$PTR array and set GLOBAL$DRIVE$NO (if a non-overlapped operation is to be started).

At this point, parameters from the operation control block are output to the DMA controller and the FDC

command phase is initiated. After completion of the command phase, a test is made to determine the type of result phase required for the current operation. If no result phase is needed, control is immediately returned to the calling program. If an immediate result phase is required, the result bytes are input from the FDC. Otherwise, the CPU waits until the OPERATION$COMPLETE flag is set (by the interrupt service procedure).

Finally, if an error is detected in the result status code (from the FDC), an FDC operation error is reported to the calling program.

Table 7. Driver Data Base

Name	Description
DRIVE$READY	A public array containing the current "ready" status of each drive.
DRIVE$STATUS$CHANGE	A public array containing a flag for each drive. The appropriate flag is set whenever the ready status of a drive changes.
OPERATION$DOCB$PTR	An internal array of pointers to the operation control block currently in progress for each drive.
OPERATIONINPROGRESS	An internal array used by the driver procedures to determine if a disk operation is in progress on a given drive.
OPERATION$COMPLETE	An internal array used by the driver procedures to determine when the execution phase of a disk operation is complete.
GLOBAL$DRIVE$NO	A data byte that records the current drive number for non-overlapped disk operations.
VALID$COMMAND	A constant flag array that indicates whether a specified FDC command code is valid.
COMMAND$LENGTH	A constant byte array specifying the number of command/parameter bytes to be transferred to the FDC during the command phase.
DRIVENOPRESENT	A constant flag array that indicates whether a drive number is encoded into an FDC command.
OVERLAP$OPERATION	A constant flag array that indicates whether an FDC command can be overlapped with other commands.
NO$RESULT	A constant flag array that is used to determine when an FDC operation does not have a result phase.
IMMED$RESULT	A constant flag array that indicates that an FDC operation has a result phase beginning immediately after the command phase is complete.
POSSIBLE$ERROR	A constant flag array that indicates if an FDC operation should be checked for an error status indication during the result phase.

Address Offset	Disk Operation Control Block (DOCB)
0	DMA$OP
1	DMA$ADDR
3	DMA$ADDR$EXT
4	DMA$COUNT
6	DISK$COMMAND (0)
7	DISK$COMMAND (1)
8	DISK$COMMAND (2)
9	DISK$COMMAND (3)
10	DISK$COMMAND (4)
11	DISK$COMMAND (5)
12	DISK$COMMAND (6)
13	DISK$COMMAND (7)
14	DISK$COMMAND (8)
15	DISK$RESULT (0)
16	DISK$RESULT (1)
17	DISK$RESULT (2)
18	DISK$RESULT (3)
19	DISK$RESULT (4)
20	DISK$RESULT (5)
21	DISK$RESULT (6)
22	MISC

Figure 2. Disk Operation Control Block (DOCB) Format

Table 8. EXECUTE$DOCB Return Status Codes

Code	Description
0	**NO ERRORS.** The specified operation was completed without error.
1	**FDC BUSY.** The requested operation cannot be started. This error occurs if an attempt is made to start an operation before the previous operation is completed.
2	**FDC ERROR.** An error was detected by the FDC during the execution phase of a disk operation. Additional error information is contained in the result data portion of the disk operation control block (DOCB.DISK$RESULT) as described in the 8272 data sheet. This error occurs whenever the 8272 reports an execution phase error (e.g., missing address mark).
3	**8272 COMMAND INTERFACE ERROR.** An 8272 interfacing error was detected during the command phase. This error occurs when the command phase of a disk operation cannot be successfully completed (e.g., incorrect setting of the DIO flag in the Main Status Register).
4	**8272 RESULT INTERFACE ERROR.** An 8272 interfacing error was detected during the result phase. This error occurs when the result phase of a disk operation cannot be successfully completed (e.g., incorrect setting of the DIO flag in the Main Status Register).
5	**INVALID FDC COMMAND.**

FDCINT

This procedure performs all interrupt processing for the 8272 interface drivers. Basically, two types of interrupts are generated by the 8272: (a) an interrupt that signals the end of a command execution phase and the beginning of the result phase and (b) an interrupt that signals the completion of an overlapped operation or the occurrence of an unexpected event (e.g., change in the drive "ready" status).

An interrupt of type (a) is indicated when the FDC BUSY flag is set (in the Main Status Register). When a type (a) interrupt is sensed, the result bytes are read from the 8272 and placed in the result portion of the disk operation control block, the appropriate OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

When an interrupt of type (b) is indicated (FDC not busy), a sense interrupt status command is issued (to the FDC). The upper two bits of the result status register (Status Register Zero—ST0) are used to determine the cause of the interrupt. The following four cases are possible:

1) Operation Complete. An overlapped operation is complete. The drive number is found in the lower two bits of ST0. The ST0 data is transferred to the active operation control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.
2) Abnormal Termination. A disk operation has abnormally terminated. The drive number is found in the lower two bits of ST0. The ST0 data is transferred to the active control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.
3) Invalid Command. The execution of an invalid command (i.e., a sense interrupt command with no interrupt pending) has been attempted. This interrupt signals the successful completion of all interrupt processing.
4) Drive Status Change. A change has occurred in the "ready" status of a disk drive. The drive number is found in the lower two bits of ST0. The DRIVE$READY flag for this disk drive is set to the new drive "ready" status and the DRIVE$STATUS$CHANGE flag for the drive is also set. In addition, if a command is currently in progress, the ST0 data is transferred to the active control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

After processing a type (b) interrupt, additional sense interrupt status commands must be issued and processed until an "invalid command" result is returned from the FDC. This action guarantees that all "hidden" interrupts are serviced.

In addition to the major driver procedures described above, a number of support procedures are required. These support routines are briefly described in the following paragraphs.

OUTPUT$CONTROLS$TO$DMA

This procedure outputs the DMA mode, the DMA address, and the DMA word count to the 8237 DMA controller. In addition, the upper four bits of the 20-bit DMA address are output to the address extension latch. Finally, the disk DMA channel is started.

OUTPUT$COMMAND$TO$FDC

This software module outputs a complete disk command to the 8272 FDC. The number of required command/parameter bytes is found in the COMMAND$LENGTH table. The appropriate bytes are output one at a time (by calls to OUTPUT$BYTE$TO$FDC) from the command portion of the disk operation control block.

INPUT$RESULT$FROM$FDC

This procedure is used to read result phase status information from the disk controller. At most, seven bytes are read. In order to read each byte, a call is made to INPUT$BYTE$FROM$FDC. When the last byte has been read, a check is made to insure that the FDC is no longer busy.

OUTPUT$BYTE$TO$FDC

This software is used to output a single command/parameter byte to the FDC. This procedure waits until the FDC is ready for a command byte and then outputs the byte to the FDC data port.

INPUT$BYTE$FROM$FDC

This procedure inputs a single result byte from the FDC. The software waits until the FDC is ready to transfer a result byte and then reads the byte from the FDC data port.

FDC$READY$FOR$COMMAND

This procedure assures that the FDC is ready to accept a command/parameter byte by performing the following three steps. First, a small time interval (more than 20 microseconds) is inserted to assure that the RQM flag has time to become valid (after the last byte transfer). Second, the master request flag (RQM) is polled until it is activated by the FDC. Finally, the DIO flag is checked to ensure that it is properly set for FDC input (from the processor).

FDC$READY$FOR$RESULT

The operation of this procedure is similar to the FDC$READY$FOR$COMMAND with the following exception. If the FDC BUSY flag (in the Main Status Register) is not set, the result phase is complete and no more data is available from the FDC. Otherwise, the procedure waits for the RQM flag and checks the DIO flag for FDC output (to the processor).

OPERATION$CLEAN$UP

This procedure is called after the execution of a disk operation that has no result phase. OPERATION$-CLEAN$UP resets the OPERATIONIN-PROGRESS flag and the GLOBAL$DRIVE$NO variable if appropriate. This procedure is also called to clean up after some disk operation errors.

Modifications for Polling Operation

To operate in the polling mode, the following modifications should be made to the previous routines:

1) The OUTPUT$CONTROLS$TO$DMA routine should be deleted.
2) In EXECUTE$DOCB, immediately prior to WAIT-FOROP$COMPLETE, a polling loop should be inserted into the code. The loop should test the RQM flag (in the Main Status Register). When RQM is set, a data byte should be written to, or read from, the 8272. The buffer address may be computed from the base address contained in DOCB.DMA$-ADDR and DOCB.DMA$ADDR$EXT. After the correct number of bytes have been transferred, an operation complete interrupt will be issued by the FDC. During data transfer in the non-DMA mode, the NON-DMA MODE flag (bit 5 of the Main Status Register) will be set. This flag will remain set for the complete execution phase. When the transfer is finished, the NON-DMA MODE flag is reset and the result phase interrupt is issued by the FDC.

5.0 8272 LOGICAL INTERFACE SOFTWARE

Appendix B of this Application Note contains a PL/M listing of an exerciser program for the 8272 drivers. This program illustrates the design of logical interface level procedures to specify disk parameters, recalibrate a drive, seek to a cylinder, format a disk, read data, and write data.

The exerciser program is written to operate a standard single-sided 8″ floppy disk drive in either the single- or double-density recording mode. Only the eight parameters listed in Table 9 must be specified. All other parameters are derived from these 8 basic variables.

Each of these logical interface procedures is described in the following paragraphs (refer to the listing in Appendix B).

SPECIFY

This procedure sets the FDC signal timing so that the FDC will interface correctly to the attached disk drive. The SPECIFY procedure requires four parameters, the step rate (SRT), head load time (HLT), head unload time (HUT), and the non-DMA mode flag (ND). This procedure builds a disk operation control block (SPE-CIFY$DOCB) and passes the control block to the FDC driver module (EXECUTE$DOCB) for execution. (Note carefully the computation required to transform the step rate (SRT) into the correct 8272 parameter byte.)

RECALIBRATE

This procedure causes the floppy disk read/write head to retract to track 0. The RECALIBRATE procedure requires only one parameter—the drive number on which the recalibrate operation is to be performed. This procedure builds a disk operation control block (RE-CALIBRATE$DOCB) and passes the control block to the FDC driver for execution.

SEEK

This procedure causes the disk read/write head (on the selected drive) to move to the desired cylinder position. The SEEK procedure is called with three parameters: drive number (DRV), head/side number (HD), and cylinder number (CYL). This software module builds a disk operation control block (SEEK$DOCB) that is executed by the FDC driver.

FORMAT

The FORMAT procedure is designed to initialize a complete floppy disk so that sectors can subsequently be read and written by system and application programs. Three parameters must be supplied to this procedure: the drive number (DRV), the recording density (DENS), and the interleave factor (INTLVE). The FORMAT procedure generates a data block (FMTBLK) and a disk operation control block (FOR-MAT$DOCB) for each track on the floppy disk (normally 77).

Table 9. Basic Disk Parameters

Name	Description
DENSITY	The recording mode (FM or MFM).
FILLER$BYTE	The data byte to be written in all sectors during formatting.
TRACKSPERDISK	The number of cylinders on the floppy disk.
BYTESPERSECTOR	The number of bytes in each disk sector. The exerciser accepts 128, 256, and 512 in FM mode, and 256, 512, and 1024 in MFM mode.
INTERLEAVE	The sector interleave factor for each disk track.
STEP$RATE	The disk drive step rate (1–16 milliseconds).
HEAD$LOAD$TIME	The disk drive head load time (2–254 milliseconds).
HEAD$UNLOAD$TIME	The head unload time (16–240 milliseconds).

The format data block specifies the four sector ID field parameters (cylinder, head, sector, and bytes per sector) for each sector on the track. The sector numbers need not be sequential; the interleave factor (INTLVE parameter) is used to compute the logical to physical sector mapping.

After both the format data block and the operation control block are generated for a given cylinder, control is passed to the 8272 drivers for execution. After the format operation is complete, a SEEK to the next cylinder is performed, a new format table is generated, and another track formatting operation is executed by the drivers. This track formatting continues until all tracks on the diskette are formatted.

In some systems, bad tracks must also be specified when a disk is formatted. For these systems, the existing FORMAT procedure should be modified to format bad tracks with a cylinder number of 0FFH.

Write

The WRITE procedure transfers a complete sector of data to the disk drive. Five parameters must be supplied to this software module: the drive number (DRV), the cylinder number (CYL), the head/side number (HD), the sector number (SEC) and the recording density (DENS). This procedure generates a disk operation control block (WRITE$DOCB) from these parameters and passes the control block to the 8272 driver for execution. When control returns to the calling program, the data has been transferred to disk.

Read

This procedure is identical to the WRITE procedure except the direction of data transfer is reversed. The READ procedure transfers a sector of data from the floppy disk to system memory.

Coping with Errors

In actual practice all logical disk interface routines would contain error processing mechanisms. (Errors have been ignored for the sake of simplicity in the exerciser programs listed in Appendix B.) A typical error recovery technique consists of a two-stage procedure. First, when an error is detected, a recalibrate operation is performed followed by a retry of the failed operation. This procedure forces the drive to seek directly to the requested cylinder (lowering the probability of a seek error) and attempts to perform the requested operation an additional time. Soft (temporary) errors caused by mechanical or electrical interference do not normally recur during the retry operation; hard errors (caused by media or drive failures), on the other hand, will continue to occur during retry operations. If, after a number of retries (approximately 10), the operation continues to fail, an error message is displayed to the system operator. This error message lists the drive number, type of operation, and failure status (from the FDC). It is the operator's responsibility to take additional action as required.

6.0 FILE SYSTEMS

The file system provides the disk I/O interface level most familiar to users of interactive microcomputer and minicomputer systems. In a file system, all data is stored in named disk areas called files. The user and applications programs need not be concerned with the exact location of a file on the disk—the disk file system automatically determines the file location from the file name. Files may be created, read, written, modified, and finally deleted (destroyed) when they are no longer needed. Each floppy disk typically contains a directory that lists all the files existing on the disk. A directory entry for a file contains information such as file name, file size, and the disk address (track and sector) of the beginning of the file.

File Allocation

File storage is actually allocated on the disk (by the file system) in fixed size areas called blocks. Normally a block is the same size as a disk sector. Files are created by finding and reserving enough unused blocks to contain the data in the file. Two file allocation methods are currently in widespread use. The first method allocates blocks (for a file) from a sequential pool of unused blocks. Thus, a file is always contained in a set of sequential blocks on the disk. Unfortunately, as files are created, updated, and deleted, these free-block pools become fragmented (separated from one another). When this fragmentation occurs, it often becomes impossible for the file system to create a file even though there is a sufficient number of free blocks on the disk. At this point, special programs must be run to "squeeze" or compact the disk, in order to re-create a single contiguous free-block pool.

The second file allocation method uses a more flexible technique in which individual data blocks may be located anywhere on the disk (with no restrictions). With this technique, a file directory entry contains the disk address of a file pointer block rather than the disk address of the first data block of the file. This file pointer block contains pointers (disk addresses) for each data block in the file. For example, the first pointer in the file pointer block contains the track and sector address of the first data block in the file; the second pointer contains the disk address of the second data block, etc.

In practice, pointer blocks are usually the same size as data blocks. Therefore, some files will require multiple pointer blocks. To accommodate this requirement without loss of flexibility, pointer blocks are linked together, that is, each pointer block contains the disk address of the following pointer block. The last pointer block of the file is signaled by an illegal disk address (e.g., track 0, sector 0 or track 0FFH, sector 0FFH).

The Intel File System

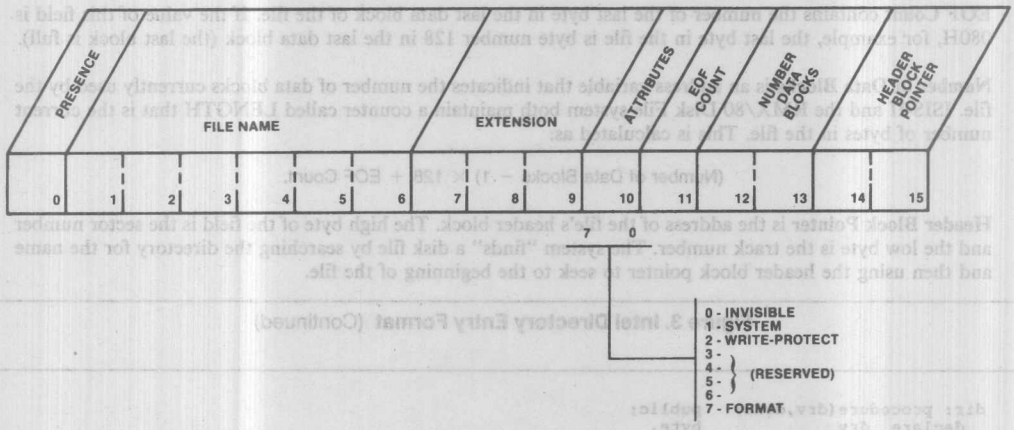
The Intel file system (described in detail in the RMX-80 Users Guide) uses the second disk file allocation method (previously discussed). In order to lower the system overhead involved in finding free data blocks, the Intel file system incorporates a free space management data structure known as a bit map. Each disk sector is represented by a single bit in the bit map. If a bit in the bit map is set to 1, the corresponding disk sector has been allocated. A zero in the bit map indicates that the corresponding sector is free. With this technique, the process of allocating or freeing a sector is accomplished by simply altering the bit map.

File names consist of a basic file name (up to six characters) and a file extension (up to three characters). The basic file name and the file extension are separated by a period (.). Examples of valid file names are: DRIV72.OBJ, XX.TMP, and FILE.CS. In addition, four file attributes are supported (see Figure 3 for attribute definitions).

The bit map and the file directory are placed on prespecified disk tracks (reserved for system use) beginning at track zero.

Disk File System Functions

Table 2 illustrates the typical functions implemented by a disk file system. As an example, the disk directory function (DIR) lists disk file information on the console display terminal. Figure 3 details the contents of a display entry in the Intel file system. The PL/M procedure outlined in Figure 4 illustrates a disk directory algorithm that displays the file name, the file attributes, and the file size (in blocks) for each file in the directory.



207885-5

Directory Entry

Presence is a flag that can contain one of three values:

000H— The file associated with this entry is present on the disk.

07FH— No file is associated with this entry; the content of the rest of the entry is undefined. The first entry with its flag set to 07FH marks the current logical end of the directory and directory searches stop at this entry.

0FFH— The file named in this entry once existed on the disk but is currently deleted. The next file added to the directory will be placed in the first entry marked 0FFH. This flag cannot, therefore, be used to (reliably) find a file that has been deleted. A value of 0FFH should be thought of as simply marking an open directory entry.

File Name is a string of up to 6 non-blank ASCII characters specifying the name of the file associated with the directory entry. If the file name is shorter than six characters, the remaining bytes contain binary zeros. For example, the name ALPHA would be stored as: 414C50484100H.

Extension is a string of up to 3 non-blank ASCII characters that specifies an extension to the file name. Extensions often identify the type of data in the file such as OBJ (object module), or PLM (PL/M source module). As with the file name, unused positions in the extension field are filled with binary zeros.

Attributes are bits that identify certain characteristics of the file. A 1 bit indicates that the file has the attribute, while a 0 bit means that the file does not have the attribute. The bit positions and their corresponding attributes are listed below (bit 0 is the low-order or rightmost bit, bit 7 is the leftmost bit):

0: **Invisible.** Files with this attribute are not listed by the ISIS-II DIR command unless the I switch is used. All system files are invisible.

1: **System.** Files with this attribute are copied to the disk in drive 1 when the S switch is specified with the ISIS-II FORMAT command.

2: **Write-Protect.** Files with this attribute cannot be opened for output or update, nor can they be deleted or renamed.

3–6: These positions are reserved for future use.

7: **Format.** Files with this attribute are treated as though they are write-protected. In addition, these files are created on a new diskette when the ISIS-II FORMAT command is issued. The system files all have the FORMAT attribute and it should not be given to any other files.

Figure 3. Intel Directory Entry Format

EOF Count contains the number of the last byte in the last data block of the file. If the value of this field is 080H, for example, the last byte in the file is byte number 128 in the last data block (the last block is full).

Number of Data Blocks is an address variable that indicates the number of data blocks currently used by the file. ISIS-II and the RMX/80 Disk File system both maintain a counter called LENGTH that is the current number of bytes in the file. This is calculated as:

$$(\text{Number of Data Blocks} - 1) \times 128 + \text{EOF Count}.$$

Header Block Pointer is the address of the file's header block. The high byte of the field is the sector number and the low byte is the track number. The system "finds" a disk file by searching the directory for the name and then using the header block pointer to seek to the beginning of the file.

Figure 3. Intel Directory Entry Format (Continued)

```
dir: procedure(drv,dens)    public;
  declare  drv              byte,
           dens             byte,
           sector           byte,
           i                byte,
           dir$ptr          byte,
           dir$entry        based rdbptr structure (presence byte,
                            file$name(6) byte,extension(3) byte,
                            attribute byte,eof$count byte,
                            data$blocks address,header$ptr address),
           size (5)         byte,

           invisible$flag   literally '1',
           system$flag      literally '2',
           protected$flag   literally '4',
           format$flag      literally '80H';

  /* The disk directory starts at cylinder 1, sector 2 */
  call seek(drv,1,0);
  do sector=2 to 26;
    call read(drv,1,0,sector,dens);
    do dir$ptr=0 to 112 by 4;
      if dir$entry.presence=7FH then return;
      if dir$entry.presence=0
        then do;
          do i=0 to 5; call co(dir$entry.file$name(i)); end;
          call co(period);
          do i=0 to 2; call co(dir$entry.extension(i)); end;
          do i=0 to 4; call co(space); end;
          call convert$to$decimal(@size,dir$entry.data$blocks);
          do i=0 to 4; call co(size(i)); end;
          If (dir$entry.attribute and invisible$flag) <> 0 then call co('I');
          If (dir$entry.attribute and system$flag) <> 0 then call co('S');
          If (dir$entry.attribute and protected$flag) <> 0 then call co('W');
          If (dir$entry.attribure and format$flag) <> 0 then call co('F');
        end;
    end;
  end;
end dir;
```

207885-6

Figure 4. Sample PL/M Directory Procedure

7.0 KEY 8272 SOFTWARE INTERFACING CONSIDERATIONS

This section contains a quick review of Key 8272 Software design features and issues. (Most items have been mentioned in other sections of this application note.) Before designing 8272 software drivers, it is advisable that the information in this section be thoroughly understood.

1. Non-DMA Transfers

In systems that operate without a DMA controller (in the polled or interrupt driven mode), the system software is responsible for counting data transfers to/from the 8272 and generating a TC signal to the FDC when the transfer is complete.

2. Processor Command/Result Phase Interface

In the command phase, the driver software must write the exact number of parameters in the exact order shown in Table 5. During the result phase, the driver must read the complete result status. For example, the Format Track command requires six command bytes and presents seven result bytes. The 8272 will not accept a new command until all result bytes are read. Note that the number of command and result bytes varies from command-to-command. **Command and result phases cannot be shortened.**

During both the command and result phases, the Main Status Register must be read by the driver before each byte of information is read from, or written to, the FDC Data Register. Before each command byte is written, DIO (bit 6) must be low (indicating a data transfer from the processor) and RQM (bit 7) must be high (indicating that the FDC is ready for data). During the result phase, DIO must be high (indicating a data transfer to the processor).

NOTE:
After the 8272 receives a command byte, the RQM flag may remain set for approximately 16 microseconds (with an 8 MHz clock). The driver should not attempt to read the Main Status Register before this time interval has elapsed; otherwise, the driver may erroneously assume that the FDC is ready to accept the next byte.

3. Sector Sizes

The 8272 does not support 128 byte sectors in the MFM (double-density) mode.

4. Drive Status Changes

The 8272 constantly polls all drives for changes in the drive ready status. This polling begins immediately following RESET. An interrupt is generated every time the FDC senses a change in the drive ready status. After reset, the FDC assumes that all drives are "not ready". If a drive is ready immediately after reset, the 8272 generates a drive status change interrupt.

5. Seek Commands

The 8272 FDC does not perform implied seeks. Before issuing a data read or write command, the read/write head must be positioned over the correct cylinder by means of an explicit seek command. If the head is not positioned correctly, a cylinder address error is generated.

6. Interrupt Processing

When the processor receives an interrupt from the FDC, the FDC may be reporting one of two distinct events:

a) The beginning of the result phase of a previously requested read, write, or scan command.

b) An asynchronous event such as a seek/recalibrate completion, an attention, an abnormal command termination, or an invalid command.

These two cases are distinguished by the FDC BUSY flag (bit 4) in the Main Status Register. If the FDC BUSY flag is high, the interrupt is of type (a). If the FDC BUSY flag is low, the interrupt was caused by an asynchronous event (b).

A single interrupt from the FDC may signal more than one of the above events. After receiving an interrupt, the processor must continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are ferreted out and serviced.

7. Skip Flag (SK)

The skip flag is used during the execution of Read Data, Read Deleted Data, Read Track, and various Scan commands. This flag permits the FDC to skip unwanted sectors on a disk track.

When performing a Read Data, Read Track, or Scan command, a high SK flag indicates that the FDC is to skip over (not transfer) any sector containing a deleted data address mark. A low SK flag indicates that the FDC is to terminate the command (after reading all the data in the sector) when a deleted data address mark is encountered.

When performing a Read Deleted Data command, a high SK flag indicates that sectors containing normal

data address marks are to be skipped. Note that this is just the opposite situation from that described in the last paragraph. When a data address mark is encountered during a Read Deleted Data command (and the SK flag is low), the FDC terminates the command after reading all the data in the sector.

8. Bad Track Maintenance

The 8272 does not internally maintain bad track information. The maintenance of this information must be performed by system software. As an example of typical bad track operation, assume that a media test determines that track 31 and track 66 of a given floppy disk are bad. When the disk is formatted for use, the system software formats physical track 0 as logical cylinder 0 (C = 0 in the command phase parameters), physical track 1 as logical track 1 (C = 1), and so on, until physical track 30 is formatted as logical cylinder 30 (C = 30). Physical track 31 is bad and should be formatted as logical cylinder FF (indicating a bad track). Next, physical track 32 is formatted as logical cylinder 31, and so on, until physical track 65 is formatted as logical cylinder 64. Next, bad physical track 66 is formatted as logical cylinder FF (another bad track marker), and physical track 67 is formatted as logical cylinder 65. This formatting continues until the last physical track (77) is formatted as logical cylinder 75. Normally, after this formatting is complete, the bad track information is stored in a prespecified area on the floppy disk (typically in a sector on track 0) so that the system will be able to recreate the bad track information when the disk is removed from the drive and reinserted at some later time.

To illustrate how the system software performs a transfer operation on a disk with bad tracks, assume that the disk drive head is positioned at track 0 and the disk described above is loaded into the drive. If a command to read track 36 is issued by an application program, the system software translates this read command into a **seek** to physical track 37 (since there is one bad track between 0 and 36, namely 31) followed by a **read** of logical cylinder 36. Thus, the cylinder parameter C is set to 37 for the Seek command and 36 for the Read Sector command.

REFERENCES

1. Intel, "8272 Single/Double Density Floppy Disk Controller Data Sheet," Intel Corporation, 1980.
2. Intel, "An Intelligent Data Base System Using the 8272," Intel Application Note," AP-116, 1981.
3. Intel, *iSBC 208 Hardware Reference Manual*, Manual Order No. 143078, Intel Corporation, 1980.
4. Intel, *RMX/80 User's Guide*, Manual Order No. 9800522, Intel Corporation, 1978.
5. Brinch Hansen, P., *Operating System Principles*, Prentice-Hall, Inc., New Jersey, 1973.
6. Flores, I., *Computer Software: Programming Systems for Digital Computers*, Prentice-Hall, Inc., New Jersey, 1965.
7. Knuth, D. E., *Fundamental Algorithms*, Addison-Wesley Publishing Company, Massachusetts, 1975.
8. Shaw, A. C., *The Logical Design of Operating Systems*, Prentice-Hall, Inc., New Jersey, 1974.
9. Watson, R. W., *Time Sharing System Design Concepts*, McGraw-Hill, Inc., New York, 1970.
10. Zarrella, J., *Operating Systems: Concepts and Principles*, Microcomputer Applications, California, 1979.

APPENDIX A
8272 FDC DEVICE DRIVER SOFTWARE

```
PL/M-86 COMPILER    8272 FLOPPY DISK CONTROLLER DEVICE DRIVERS

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE DRIVERS
OBJECT MODULE PLACED IN :F1:driv72.OBJ
COMPILER INVOKED BY:  plm86 :F1:driv72.p86 DEBUG


                $title('8272 floppy disk controller device drivers')
                $nointvector
                $optimize(2)
                $large
   1            drivers: do;
   2    1       declare
                  /* floppy disk port definitions */
                  fdc$status$port       literally '30H',        /* 8272 status port */
                  fdc$data$port         literally '31H';        /* 8272 data port */

   3    1       declare
                  /* floppy disk commands */
                  sense$int$status      literally '08H';

   4    1       declare
                  /* interrupt definitions */
                  fdc$int$level         literally '33';     /* fdc interrupt level */

   5    1       declare
                  /* return status and error codes */
                  error                 literally '0',
                  ok                    literally '1',
                  complete              literally '3',
                  false                 literally '0',
                  true                  literally '1',
                  error$in              literally 'not',
                  propagate$error       literally 'return error',

                  stat$ok               literally '0',          /* fdc operation completed without errors */
                  stat$busy             literally '1',          /* fdc is busy, operation cannot be started */
                  stat$error            literally '2',          /* fdc operation error */
                  stat$command$error    literally '3',          /* fdc not ready for command phase */
                  stat$result$error     literally '4',          /* fdc not ready for result phase */
                  stat$invalid          literally '5';          /* invalid fdc command */

   6    1       declare
                  /* masks */
                  busy$mask             literally '10H',
                  DIO$mask              literally '40H',
                  RQM$mask              literally '80H',
                  seek$mask             literally '0FH',
                  result$error$mask     literally '0C0H',
                  result$drive$mask     literally '03H',
                  result$ready$mask     literally '08H';

   7    1       declare
                  /* drive numbers */
                  max$no$drives         literally '3',
                  fdc$general           literally '4';

   8    1       declare
                  /* miscellaneous control */
                  any$drive$seeking     literally '((input(fdc$status$port) and seek$mask) <> 0)',
                  command$code          literally '(docb.disk$command(0) and 1FH)',
                  DIO$set$for$input     literally '((input(fdc$status$port) and DIO$mask)=0)',
                  DIO$set$for$output    literally '((input(fdc$status$port) and DIO$mask)<>0)',
                  extract$drive$no      literally '(docb.disk$command(1) and 03H)',
                  fdc$busy              literally '((input(fdc$status$port) and busy$mask) <> 0)',
                  no$fdc$error          literally 'possible$error(command$code) and ((docb.disk$result(0)
                                                 and result$error$mask) = 0)',
                  wait$for$op$complete  literally 'do while not operation$complete(drive$no); end',
                  wait$for$RQM          literally 'do while (input(fdc$status$port) and RQM$mask) = 0; end;';

   9    1       declare
                  /* structures */
                  docb$type             literally               /* disk operation control block */
                        '(dma$op byte,dma$addr word, dma$addr$ext byte,dma$count word,
                          disk$command(9) byte,disk$result(7) byte,misc byte)';

                $eject
  10    1       declare
                  drive$status$change(4) byte public,           /* when set - indicates that drve status changed */
                  drive$ready(4) byte public;                   /* current status of drives */
```

207885-7

```
11   1      declare
              operation$in$progress(5) byte,          /* internal flags for operation with multiple drives */
              operation$complete(5) byte,             /* fdc execution phase completed */
              operation$docb$ptr(5) pointer,          /* pointers for operations in progress */
              interrupt$docb structure docb$type,     /* temporary docb for interrupt processing */
              global$drive$no byte;                   /* drive number of non-overlapped operation
                                                         in progress - if any */

12   1      declare
              /* internal vectors that contain command operational information */
              no$result(32) byte                      /* no result phase to command */
                data(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
              immed$result(32) byte                   /* immediate result phase for command */
                data(0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
              overlap$operation(32) byte              /* command permits overlapped operation of drvies */
                data(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
              drive$no$present(32) byte               /* drive number present in command information */
                data(0,0,1,0,1,1,1,1,0,1,1,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0),
              possible$error(32) byte                 /* determines if command can return with an error */
                data(0,0,1,0,0,1,1,1,1,1,1,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0),
              command$length(32) byte                 /* contains number of command bytes for each command */
                data(0,0,9,3,2,9,9,2,1,9,2,0,9,6,0,3,0,9,0,0,0,0,0,0,0,9,0,0,0,0,9,0,0),
              valid$command(32) byte                  /* flags invalid command codes */
                data(0,0,1,1,1,1,1,1,1,1,1,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0);

              $eject

              /**** initialization for the 8272 fdc driver software.  This procedure must
                    be called prior to execution of any driver software.    ****/

13   1      initialize$drivers: procedure public;
              /* initialize 8272 drivers */
14   2        declare drv$no byte;

15   2        do drv$no=0 to max$no$drives;
16   3          drive$ready(drv$no)=false;
17   3          drive$status$change(drv$no)=false;
18   3          operation$in$progress(drv$no)=false;
19   3          operation$complete(drv$no)=false;
20   3        end;

21   2        operation$in$progress(fdc$general)=false;
22   2        operation$complete(fdc$general)=false;
23   2        global$drive$no=0;

24   2      end initialize$drivers;

              /**** wait until the 8272 fdc is ready to receive command/parameter bytes
                    in the command phase.  The 8272 is ready to receive command bytes
                    when the RQM flag is high and the DIO flag is low.    ****/

25   1      fdc$ready$for$command: procedure byte;

              /* wait for valid flag settings in status register */
26   2        call time(1);

              /* wait for "master request" flag */
27   2        wait$for$RQM;

              /* check data direction flag */
30   2        if DIO$set$for$input
                then return ok;
32   2          else return error;

33   2      end fdc$ready$for$command;


              /**** wait until the 8272 fdc is ready to return data bytes in the result
                    phase.  The 8272 is ready to return a result byte when the RQM and DIO
                    flags are both high.  The busy flag in the main status register will
                    remain set until the last data byte of the result phase has been read
                    by the processor.    ****/

34   1      fdc$ready$for$result: procedure byte;

              /* wait for valid settings in status register */
35   2        call time(1);

              /* result phase has ended when the 8272 busy flag is reset */
36   2        if not fdc$busy
                then return complete;
```

207885-8

```
                  /* wait for "master request" flag */
38    2           wait$for$RQM;

                  /* check data direction flag */
41    2           if DIO$set$for$output
                    then return ok;
43    2           else return error;

44    2         end fdc$readv$for$result;


                /**** output a single command/parameter byte to the 8272 fdc.  The "data$byte"
                      parameter is the byte to be output to the fdc.     ****/

45    1         output$byte$to$fdc: procedure(data$byte) byte;
46    2           declare data$byte byte;

                  /* check to see if fdc is ready for command */
47    2           if not fdc$readv$for$command
                    then propagate$error;

49    2           output(fdc$data$port)=data$byte;

50    2           return ok;
51    2         end output$byte$to$fdc;


                /**** input a single result byte from the 8272 fdc.  The "data$byte$ptr"
                      parameter is a pointer to the memory location that is to contain
                      the input byte.    ****/

52    1         input$byte$from$fdc: procedure(data$byte$ptr) byte;
53    2           declare data$byte$ptr pointer;
54    2           declare
                    data$byte based data$byte$ptr byte,
                    status byte;

                  /* check to see if fdc is ready */
55    2           status=fdc$ready$for$result;
56    2           if error$in status
                    then propagate$error;

                  /* check for result phase complete */
58    2           if status=complete
                    then return complete;

60    2           data$byte=input(fdc$data$port);
61    2           return ok;
62    2         end input$byte$from$fdc;

                $eject

                /**** output the dma mode, the dma address, and the dma word count to the
                      8237 dma controller.  Also output the high order four bits of the
                      address to the address extension latch.  Finally, start the disk
                      dma channel.  The "docb$ptr" parameter is a pointer to the appropriate
                      disk operation control block.    ****/

63    1         output$controls$to$dma: procedure(docb$ptr);
64    2           declare docb$ptr pointer;
65    2           declare docb based docb$ptr structure docbtype;

66    2           declare
                    /* dma port definitions */
                    dma$upper$addr$port   literally '10H',      /* upper 4 bits of current address */
                    dma$disk$addr$port    literally '00H',      /* current address port */
                    dma$disk$word$count   literally '01H',      /* word count port */
                    dma$command$port      literally '08H',      /* command port */
                    dma$mode$port         literally '0BH',      /* mode port */
                    dma$mask$sr$port      literally '0AH',      /* mask set/reset port */
                    dma$clear$ff$port     literally '0CH',      /* clear first/last flip-flop port */
                    dma$master$clear$port literally '0DH',      /* dma master clear port */
                    dma$mask$port         literally '0FH',      /* parallel mask set port*/

                    dma$disk$chan$start   literally '00H',      /* dma mask to start disk channel */
                    dma$extended$write    literally 'shl(1,5)', /* extended write flag */
                    dma$single$transfer   literally 'shl(1,6)'; /* single transfer flag */

67    2           if docb.dma$op < 3
                    then do;
                      /* set dma mode and clear first/last flip-flop */
69    3               output(dma$mode$port)=shl(docb.dma$op,2) or 40H;
70    3               output(dma$clear$ff$port)=0;
```

207885-9

```
                        /* set dma address */
 71   3                 output(dma$disk$addr$port)=low(docb.dma$addr);
 72   3                 output(dma$disk$addr$port)=high(docb.dma$addr);
 73   3                 output(dma$upper$addr$port)=docb.dma$addr$ext;

                        /* output disk transfer word count to dma controller */
 74   3                 output(dma$disk$word$count)=low(docb.dma$count);
 75   3                 output(dma$disk$word$count)=high(docb.dma$count);

                        /* start dma channel 0 for fdc */
 76   3                 output(dma$mask$sr$port)=dma$disk$chan$start;
 77   3               end;

 78   2     end output$controls$to$dma;


                      /**** output a high-level disk command to the 8272 fdc.  The number of bytes
                            required for each command is contained in the "command$length" table.
                            The "docb$ptr" parameter is a pointer to the appropriate disk operation
                            control block.    ****/

 79   1     output$command$to$fdc: procedure(docb$ptr) byte;
 80   2       declare docb$ptr pointer;

 81   2       declare
                 docb based docb$ptr structure docb$type,
                 cmd$byte$no byte;

 82   2       disable;

                 /* output all command bytes to the fdc */
 83   2       do cmd$byte$no=0 to command$length(command$code)-1;
 84   3         if error$in output$byte$to$fdc(docb.disk$command(cmd$byte$no))
                    then do; enable; propagate$error; end;
 89   3       end;

 90   2       enable;
 91   2       return ok;
 92   2     end output$command$to$fdc;


                      /**** input the result data from the 8272 fdc during the result phase (after
                            command execution).  The "docb$ptr" parameter is a pointer to the
                            appropriate disk operation control block.    ****/

 93   1     input$result$from$fdc: procedure(docb$ptr) byte;
 94   2       declare docb$ptr pointer;
 95   2       declare
                 docb based docb$ptr structure docb$type,
                 result$byte$no byte,
                 temp byte,
                 status byte;

 96   2       disable;

 97   2       do result$byte$no=0 to 7;
 98   3         status=input$byte$from$fdc(@temp);
 99   3         if error$in status
                    then do; enable; propagate$error; end;
104   3         if status=complete
                    then do; enable; return ok; end;
109   3         docb.disk$result(result$byte$no)=temp;
110   3       end;

111   2       enable;
112   2       if fdc$busy
                 then return error;
114   2       else return ok;
115   2     end input$result$from$fdc;


                      /**** cleans up after the execution of a disk operation that has no result
                            phase.  The procedure is also used after some disk operation errors.
                            "drv" is the drive number, and "cc" is the command code for the
                            disk operation.    ****/

116   1     operation$clean$up: procedure(drv,cc);
117   2       declare (drv,cc) byte;

118   2       disable;
119   2       operation$in$progress(drv)=false;
```

207885-10

```
120   2       if not overlap$operation(cc)
                then global$drive$no=0;
122   2       enable;

123   2     end operation$clean$up;

            $eject

            /**** execute the disk operation control block specified by the pointer
                  parameter "docb$ptr".  The "status$ptr" parameter is a pointer to
                  a byte variable that is to contain the status of the requested
                  operation when it has been completed.  Six status conditions are
                  possible on return:

                    0    The specified operation was completed without error.
                    1    The fdc is busy and the requested operation cannot be started.
                    2    Fdc error (further information is contained in the result
                         storage portion of the disk operation control block - as
                         described in the 8272 data sheet).
                    3    Transfer error during output of the command bytes to the fdc.
                    4    Transfer error during input of the result bytes from the fdc.
                    5    Invalid fdc command.    ****/

124   1     execute$docb: procedure(docb$ptr,status$ptr) public;
            /* execute a disk operation control block */

125   2       declare docb$ptr pointer, status$ptr pointer;
126   2       declare
                docb based docb$ptr structure docb$type,
                status based status$ptr byte,
                drive$no byte;

                /* check command validity */
127   2       if not valid$command(command$code)
                then do; status=stat$invalid; return; end;

                /* determine if command has a drive number field - if not, set the drive
                   number for a general fdc command */
132   2       if drive$no$present(command$code)
                then drive$no=extract$drive$no;
134   2         else drive$no=fdc$general;

                /* an overlapped operation can not be performed if the fdc is busy */
135   2       if overlap$operation(command$code) and fdc$busy
                then do; status=stat$busy; return; end;

                /* for a non-overlapped operation, check fdc busy or any drive seeking */
140   2       if not overlap$operation(command$code) and (fdc$busy or any$drive$seeking)
                then do; status=stat$busy; return; end;

                /* check for drive operation in progress - if none, set flag and start operation */
145   2       disable;
146   2       if operation$in$progress(drive$no)
                then do; enable; status=stat$busy; return; end;
152   2         else operation$in$progress(drive$no)=true;

                /* at this point, an fdc operation is about to begin, so:
                   1. reset the operation complete flag
                   2. set the docb pointer for the current operation
                   3. if this is not an overlapped operation, set the global drive
                      number for the subsequent result phase interrupt.  */
153   2       operation$complete(drive$no)=0;
154   2       operation$docb$ptr(drive$no)=docb$ptr;

155   2       if not overlap$operation(command$code)
                then global$drive$no=drive$no+1;
157   2       enable;

158   2       call output$controls$to$dma(docb$ptr);
159   2       if error$in output$command$to$fdc(docb$ptr)
                then do;
161   3               call operation$clean$up(drive$no,command$code);
162   3               status=stat$command$error;
163   3               return;
164   3             end;

                /* return immediately if the command has no result phase or completion interrupt - specify */
165   2       if no$result(command$code)
                then do;
167   3               call operation$clean$up(drive$no,command$code);
168   3               status=stat$ok;
169   3               return;
170   3             end;
```

207885-11

```
171   2        if immed$result(command$code)
                 then do;
173   3          if error$in input$result$from$fdc(docb$ptr)
                   then do;
175   4            call operation$clean$up(drive$no,command$code);
176   4            status=stat$result$error;
177   4            return;
178   4          end;
179   3        end;
180   2        else do;
181   3          wait$for$op$complete;
183   3          if docb.misc = error
                   then do; status=stat$result$error; return; end;
188   3        end;

189   2      if no$fdc$error
                 then status=stat$ok;
191   2        else status=stat$error;

192   2    end execute$docb;

           $eject

           /**** copy disk command results from the interrupt control block to the
                 currently active disk operation control block if a disk operation is
                 in progress.    ****/

193   1    copy$int$result: procedure(drv);
194   2      declare drv byte;
195   2      declare
                 i byte,
                 docb$ptr pointer,
                 docb based docb$ptr structure docb$type;

196   2      if operation$in$progress(drv)
                 then do;
198   3          docb$ptr=operation$docb$ptr(drv);
199   3          do i=1 to 6; docb.disk$result(i)=interrupt$docb.disk$result(i); end;
202   3          docb.misc=ok;
203   3          operation$in$progress(drv)=false;
204   3          operation$complete(drv)=true;
205   3        end;

206   2    end copy$int$result;

           /**** interrupt processing for 8272 fdc drivers.  Basically, two types of
                 interrupts are generated by the 8272: (a)when the execution phase of
                 an operation has been completed, an interrupt is generated to signal
                 the beginning of the result phase (the fdc busy flag is set
                 when this interrupt is received), and (b) when an overlapped operation
                 is completed or an unexpected interrupt is received (the fdc busy flag
                 is not set when this interrupt is received).

                 When interrupt type (a) is received, the result bytes from the operation
                 are read from the 8272 and the operation complete flag is set.

                 When an interrupt of type (b) is received, the interrupt result code is
                 examined to determine which of the following four actions are indicated:

                   1. An overlapped option (recalibrate or seek) has been completed.  The
                      result data is read from the 8272 and placed in the currently active
                      disk operation control block.
                   2. An abnormal termination of an operation has occurred.  The result
                      data is read and placed in the currently active disk operation
                      control block.
                   3. The execution of an invalid command has been attempted.  This
                      signals the successful completion of all interrupt processing.
                   4. The ready status of a drive has changed.  The "drive$ready" and
                      "drive$ready$status" change tables are updated.  If an operation
                      is currently in progress on the affected drive, the result data
                      is placed in the currently active disk operation control block.

                 After an interrupt is processed, additional sense interrupt status commands
                 must be issued and processed until an invalid command result is returned
                 from the fdc.  This action guarantees that all "hidden" interrupts
                 are serviced.    ****/
```

207885-12

```
207   1      fdcint: procedure public interrupt fdc$int$level;
208   2        declare
                 invalid byte,
                 drive$no byte,
                 docb$ptr pointer,
                 docb based docb$ptr structure docb$type;

209   2        declare
                 /* interrupt port definitions */
                 ocw2                    literally '70H',
                 nseoi                   literally 'shl(1,5)';

210   2        declare
                 /* miscellaneous flags */
                 result$code             literally 'shr(interrupt$docb.disk$result(0) and result$error$mask,6)',
                 result$drive$ready      literally '((interrupt$docb.disk$result(0) and result$ready$mask) = 0)',
                 extract$result$drive$no literally '(interrupt$docb.disk$result(0) and result$drive$mask)',
                 end$of$interrupt        literally 'output(ocw2)=nseoi';

               /* if the fdc is busy when an interrupt is received, then the result
                  phase of the previous non-overlapped operation has begun */
211   2          if fdc$busy
                   then do;
                     /* process interrupt if operation in progress */
213   3              if global$drive$no <> 0
                       then do;
215   4                  docb$ptr=operation$docb$ptr(global$drive$no-1);
216   4                  if error$in input$result$from$fdc(docb$ptr)
                           then docb.misc=error;
                           else docb.misc=ok;
218   4                  operation$in$progress(global$drive$no-1)=false;
219   4                  operation$complete(global$drive$no-1)=true;
220   4                  global$drive$no=0;
221   4                end;
222   3              end;

               /* if the fdc is not busy, then either an overlapped operation has been
                  completed or an unexpected interrupt has occurred (e.g., drive status
                  change) */
223   2          else do;
224   3            invalid=false;
225   3            do while not invalid;

                   /* perform a sense interrupt status operation - if errors are detected,
                      in the actual fdc interface, interrupt processing is discontinued */
226   4              if error$in output$byte$to$fdc(sense$int$status) then go to ignore;
228   4              if error$in input$result$from$fdc(@interrupt$docb) then go to ignore;

230   4              do case result$code;

                       /* case 0 - operation complete */
231   5                do;
232   6                  drive$no=extract$result$drive$no;
233   6                  call copy$int$result(drive$no);
234   6                end;

                       /* case 1 - abnormal termination */

235   5                do;
236   6                  drive$no=extract$result$drive$no;
237   6                  call copy$int$result(drive$no);
238   6                end;

                       /* case 2 - invalid command */
239   5                invalid=true;

                       /* case 3 - drive ready change */
240   5                do;
241   6                  drive$no=extract$result$drive$no;
242   6                  call copy$int$result(drive$no);
243   6                  drive$status$change(drive$no)=true;
244   6                  if result$drive$ready
                             then drive$ready(drive$no)=true;
                             else drive$ready(drive$no)=false;
246   6                end;
247   5              end;
248   4            end;
249   3          end;

250   2        ignore: end$of$interrupt;
251   2      end fdcint;

252   1      end drivers;
MODULE INFORMATION:
     CODE AREA SIZE     = 0615H    1557D
     CONSTANT AREA SIZE = 0000H       0D
     VARIABLE AREA SIZE = 0050H      80D
     MAXIMUM STACK SIZE = 0032H      50D
     564 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

207885-13

207885-14

APPENDIX B
8272 FDC EXERCISER PROGRAM

```
PL/M-86 COMPILER    8272 FLOPPY DISK DRIVER EXERCISE PROGRAM

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE RUN72
OBJECT MODULE PLACED IN :F1:run72.OBJ
COMPILER INVOKED BY:  plm86 :F1:run72.p86 DEBUG


                $title ('8272 floppy disk driver exercise program')
                $nointvector
                $optimize(2)
                $large
   1            run72: do;

   2   1          declare
                    docb$type               literally           /* disk operation control block */
                      '(dma$op byte,dma$addr word,dma$addr$ext byte,dma$count word,
                      disk$command(9) byte,disk$result(7) byte,misc byte)';

   3   1          declare
                    /* 8272 fdc commands */
                    fm                      literally '0',
                    mfm                     literally '1',
                    dma$mode                literally '0',
                    non$dma$mode            literally '1',
                    recalibrate$command     literally '7',
                    specify$command         literally '3',
                    read$command            literally '6',
                    write$command           literally '5',
                    format$command          literally '0DH',
                    seek$command            literally '0FH';

   4   1          declare
                    dma$verify              literally '0',
                    dma$read                literally '1',
                    dma$write               literally '2',
                    dma$noop                literally '3';

   5   1          declare
                    /* disk operation control blocks */
                    format$docb             structure docb$type,
                    seek$docb               structure docb$type,
                    recalibrate$docb        structure docb$type,
                    specify$docb            structure docb$type,
                    read$docb               structure docb$type,
                    write$docb              structure docb$type;

   6   1          declare
                    step$rate               byte,
                    head$load$time          byte,
                    head$unload$time        byte,
                    filler$byte             byte,
                    operation$status        byte,
                    interleave              byte,
                    format$gap              byte,
                    read$write$gap          byte,
                    index                   byte,
                    drive                   byte,
                    density                 byte,
                    multitrack              byte,
                    sector                  byte,
                    cylinder                byte,
                    head                    byte,       /* disk drive head */
                    tracks$per$disk         byte,
                    sectors$per$track       byte,
                    bytes$per$sector$code   byte,
                    bytes$per$sector        word;       /* number of bytes in a sector on the disk */

   7   1          declare
                    /* read and write buffers */
                    fmtblk(104)             byte public,
                    wrbuf(1024)             byte public,
                    rdbuf(1024)             byte public;

   8   1          declare
                    /* disk format initialization tables */
                    sec$trk$table(3)        byte data(26,15,8),
                    fmt$gap$table(8)        byte data(1BH,2AH,3AH,0,0,36H,54H,74H),
                    rd$wr$gap$table(8)      byte data(07H,0EH,1BH,0,0,0EH,1BH,35H);
```

207885-15

```
 9   1        declare
               /* external pointer tables and interrupt vector */
               rdbptr(2)                word external,
               wrbptr(2)                word external,
               fbptr(2)                 word external,
               intptr(2)                word external,
               intvec(80H)              word external;


10   1        execute$docb: procedure(docb$ptr,status$ptr) external;
11   2          declare docb$ptr pointer, status$ptr pointer;
12   2        end execute$docb;

13   1        initialize$drivers: procedure external;
14   2        end initialize$drivers;

               $eject

               /**** specify step rate ("srt"), head load time ("hlt"), head unload time ("hut"),
                     and dma or non-dma operation ("nd").  ****/

15   1          specify: procedure(srt,hlt,hut,nd);
16   2            declare (srt,hlt,hut,nd) byte;

17   2            specify$docb.dma$op=dma$noop;
18   2            specify$docb.disk$command(0)=specify$command;
19   2            specify$docb.disk$command(1)=shl((not srt)+1,4) or shr(hut,4);
20   2            specify$docb.disk$command(2)=(hlt and 0FEH) or (nd and 1);
21   2            call execute$docb(@specify$docb,@operation$status);

22   2          end specify;


               /****  recalibrate disk drive
                     8272 automatically steps out until the track 0 signal is activated
                     by the disk drive.   ****/

23   1          recalibrate: procedure(drv);
24   2            declare drv byte;

25   2            recalibrate$docb.dma$op=dma$noop;
26   2            recalibrate$docb.disk$command(0)=recalibrate$command;
27   2            recalibrate$docb.disk$command(1)=drv;
28   2            call execute$docb(@recalibrate$docb,@operation$status);

29   2          end recalibrate;


               /**** seek drive "drv", head (side) "hd" to cylinder "cyl".   ****/

30   1          seek: procedure(drv,cyl,hd);
31   2            declare (drv,cyl,hd) byte;

32   2            seek$docb.dma$op=dma$noop;
33   2            seek$docb.disk$command(0)=seek$command;
34   2            seek$docb.disk$command(1)=drv or shl(hd,2);
35   2            seek$docb.disk$command(2)=cyl;
36   2            call execute$docb(@seek$docb,@operation$status);

37   2          end seek;


               /**** format a complete side ("head") of a single floppy disk in drive "drv".  The density,
                     (single or double) is specified by flag "dens".   ****/

38   1          format: procedure(drv,dens,intlve);
                 /*  format disk */
39   2            declare (drv,dens,intlve) byte;
40   2            declare physical$sector byte;

41   2            call recalibrate(drv);
42   2            do cylinder=0 to tracks$per$disk-1;
                     /* set sector numbers in format block to zero before computing interleave */

43   3              do physical$sector=1 to sectors$per$track; fmtblk((physical$sector-1)*4+2)=0; end;
                     /* physical sector 1 equals logical sector 1 */
46   3              physical$sector=1;

                     /* assign interleaved sectors */
47   3              do sector=1 to sectors$per$track;
48   4                index=(physical$sector-1)*4;
```

207885-16

```
                        /* change sector and index if sector has already been assigned */
 49   4                 do while fmtblk(index+2) <> 0; index=index+4; physical$sector=physical$sector+1; end;

                        /* set cylinder, head, sector, and size code for current sector into table */
 53   4                 fmtblk(index)=cylinder;
 54   4                 fmtblk(index+1)=head;
 55   4                 fmtblk(index+2)=sector;
 56   4                 fmtblk(index+3)=bytes$per$sector$code;

                        /* update physical sector number by interleave */
 57   4                 physical$sector=physical$sector+intlve;
 58   4                 if physical$sector > sectors$per$track
                             then physical$sector=physical$sector-sectors$per$track;
 60   4               end;

                      /* seek to next cylinder */
 61   3               call seek(drv,cylinder,head);

                      /* set up format control block */
 62   3               format$docb.dma$op=dma$write;
 63   3               format$docb.dma$addr=fbptr(0)+shl(fbptr(1),4);
 64   3               format$docb.dma$addr$ext=0;
 65   3               format$docb.dma$count=sectors$per$track*4-1;
 66   3               format$docb.disk$command(0)=format$command or shl(dens,6);
 67   3               format$docb.disk$command(1)=drv or shl(head,2);
 68   3               format$docb.disk$command(2)=bytes$per$sector$code;
 69   3               format$docb.disk$command(3)=sectors$per$track;
 70   3               format$docb.disk$command(4)=format$gap;
 71   3               format$docb.disk$command(5)=filler$byte;
 72   3               call execute$docb(@format$docb,@operation$status);
 73   3             end;

 74   2         end format;


                  /**** write sector "sec" on drive "drv" at head "hd" and cylinder "cyl".  The
                        disk recording density is specified by the "dens" flag.  Data is expected to be
                        in the global write buffer ("wrbuf").    ****/

 75   1         write: procedure(drv,cyl,hd,sec,dens);
 76   2           declare (drv,cyl,hd,sec,dens) byte;

 77   2           write$docb.dma$op=dma$write;
 78   2           write$docb.dma$addr=wrbptr(0)+shl(wrbptr(1),4);
 79   2           write$docb.dma$addr$ext=0;
 80   2           write$docb.dma$count=bytes$per$sector-1;
 81   2           write$docb.disk$command(0)=write$command or shl(dens,6) or shl(multitrack,7);
 82   2           write$docb.disk$command(1)=drv or shl(hd,2);
 83   2           write$docb.disk$command(2)=cyl;
 84   2           write$docb.disk$command(3)=hd;
 85   2           write$docb.disk$command(4)=sec;
 86   2           write$docb.disk$command(5)=bytes$per$sector$code;
 87   2           write$docb.disk$command(6)=sectors$per$track;
 88   2           write$docb.disk$command(7)=read$write$gap;
 89   2           if bytes$per$sector$code = 0
                    then write$docb.disk$command(8)=bytes$per$sector;
 91   2                else write$docb.disk$command(8)=0FFH;
 92   2           call execute$docb(@write$docb,@operation$status);

 93   2         end write;


                  /**** read sector "sec" on drive "drv" at head "hd" and cylinder "cyl".  The
                        disk recording density is defined by the "dens" flag.  Data is read into
                        the global read buffer ("rdbuf").    ****/

 94   1         read: procedure(drv,cyl,hd,sec,dens);
 95   2           declare (drv,cyl,hd,sec,dens) byte;

 96   2           read$docb.dma$op=dma$read;
 97   2           read$docb.dma$addr=rdbptr(0)+shl(rdbptr(1),4);
 98   2           read$docb.dma$addr$ext=0;
 99   2           read$docb.dma$count=bytes$per$sector-1;
100   2           read$docb.disk$command(0)=read$command or shl(dens,6) or shl(multitrack,7);
101   2           read$docb.disk$command(1)=drv or shl(hd,2);
102   2           read$docb.disk$command(2)=cyl;
103   2           read$docb.disk$command(3)=hd;
104   2           read$docb.disk$command(4)=sec;
105   2           read$docb.disk$command(5)=bytes$per$sector$code;
106   2           read$docb.disk$command(6)=sectors$per$track;
107   2           read$docb.disk$command(7)=read$write$gap;
```

207885-17

```
108   2          if bytes$per$sector$code = 0
                   then read$docb.disk$command(8)=bytes$per$sector;
110   2          else read$docb.disk$command(8)=0FFH;
111   2        call execute$docb(@read$docb,@operation$status);

112   2      end read;

$eject

        /**** initialize system by setting up 8237 dma controller and 8259A interrupt
              controller.    ****/

113   1      initialize$system: procedure;
114   2        declare
                 /* I/O ports */
                 dma$disk$addr$port       literally '00H',      /* current address port */
                 dma$disk$word$count$port literally '01H',      /* word count port */
                 dma$command$port         literally '08H',      /* command port */
                 dma$mode$port            literally '0BH',      /* mode port */
                 dma$mask$sr$port         literally '0AH',      /* mask set/reset port */
                 dma$clear$ff$port        literally '0CH',      /* clear first/last flip-flop port */
                 dma$master$clear$port    literally '0DH',      /* dma master clear port */
                 dma$mask$port            literally '0FH',      /* parallel mask set port*/
                 dma$c1$addr$port         literally '02H',
                 dma$c1$word$count$port   literally '03H',
                 dma$c2$addr$port         literally '04H',
                 dma$c2$word$count$port   literally '05H',
                 dma$c3$addr$port         literally '06H',
                 dma$c3$word$count$port   literally '07H',
                 icw1                     literally '70H',
                 icw2                     literally '71H',
                 icw4                     literally '71H',
                 ocw1                     literally '71H',
                 ocw2                     literally '70H',
                 ocw3                     literally '70H';

115   2        declare
                 /* misc masks and literals */
                 dma$extended$write       literally 'shl(1,5)',   /* extended write flag */
                 dma$single$transfer      literally 'shl(1,6)',   /* single transfer flag */
                 dma$disk$mode            literally '40H',
                 dma$c1$mode              literally '41H',
                 dma$c2$mode              literally '42H',
                 dma$c3$mode              literally '43H',
                 mode$8088                literally '1',
                 interrupt$base           literally '20H',
                 single$controller        literally 'shl(1,1)',
                 level$sensitive          literally 'shl(1,3)',
                 control$word$4$required  literally '1',
                 base$icw1                literally '10H',
                 mask$all                 literally '0FFH',
                 disk$interrupt$mask      literally '1';

116   2          output(dma$master$clear$port)=0;           /* master reset */
117   2          output(dma$mode$port)=dma$extended$write;  /* set dma command mode */

                 /* set all dma registers to valid values */
118   2          output(dma$mask$port)=mask$all;            /* mask all channels */

                 /* set all addresses to zero */
119   2          output(dma$clear$ff$port)=0;               /* reset first/last flip-flop */
120   2          output(dma$disk$addr$port)=0;
121   2          output(dma$disk$addr$port)=0;
122   2          output(dma$c1$addr$port)=0;
123   2          output(dma$c1$addr$port)=0;
124   2          output(dma$c2$addr$port)=0;
125   2          output(dma$c2$addr$port)=0;
126   2          output(dma$c3$addr$port)=0;
127   2          output(dma$c3$addr$port)=0;

                 /* set all word counts to valid values */
128   2          output(dma$clear$ff$port)=0;               /* reset first/last flip-flop */
129   2          output(dma$disk$word$count$port)=1;
130   2          output(dma$disk$word$count$port)=1;
131   2          output(dma$c1$word$count$port)=1;
132   2          output(dma$c1$word$count$port)=1;
133   2          output(dma$c2$word$count$port)=1;
134   2          output(dma$c2$word$count$port)=1;
135   2          output(dma$c3$word$count$port)=1;
136   2          output(dma$c3$word$count$port)=1;
```

207885-18

```
                    /* initialize all dma channel modes */
137   2             output(dma$mode$port)=dma$disk$mode;
138   2             output(dma$mode$port)=dma$c1$mode;
139   2             output(dma$mode$port)=dma$c2$mode;
140   2             output(dma$mode$port)=dma$c3$mode;

                    /* initialize 8259A interrupt controller */
141   2             output(icw1)=single$controller or level$sensitive or control$word4$required or base$icw1;
142   2             output(icw2)=interrupt$base;
143   2             output(icw4)=mode$8088;                    /* set 8088 interrupt mode */
144   2             output(ocw1)=not disk$interrupt$mask;      /* mask all interrupts except disk */

                    /* initialize interrupt vector for fdc */
145   2             intvec(40H)=intptr(0);
146   2             intvec(41H)=intptr(1);

147   2           end initialize$system;
                  |
                $eject

                /**** main program:   first format disk (all tracks on side (head) 0.  Then
                      read each sector on every track of the disk forever.    ****/

148   1           declare drive$ready(4) byte external;

                  /* disable until interrupt vector setup and initialization complete */
149   1           disable;

                  /* set initial floppy disk parameters */
150   1           density=mfm;                               /* double-density */
151   1           head=0;                                    /* single sided */
152   1           multitrack=0;                              /* no multitrack operation */
153   1           filler$byte=55H;                           /* for format */
154   1           tracks$per$disk=77;                        /* normal floppy disk drive */
155   1           bytes$per$sector=1024;                     /* 1024 bytes in each sector */
156   1           interleave=6;                              /* set track interleave factor */
157   1           step$rate=11;                              /* 10ms for SA800 plus 1 for uncertainty */
158   1           head$load$time=40;                         /* 40ms head load for SA800 */
159   1           head$unload$time=240;                      /* keep head loaded as long as possible */

                  /* derive dependent parameters from those above */
160   1           bytes$per$sector$code=shr(bytes$per$sector,7);
161   1           do index=0 to 3;
162   2             if (bytes$per$sector$code and 1) <> 0
                      then do; bytes$per$sector$code=index; go to donebc; end;
167   2             else bytes$per$sector$code=shr(bytes$per$sector$code,1);
168   2           end;

169   1           donebc:
                  sectors$per$track=sec$trk$table(bytes$per$sector$code-density);
170   1           format$gap=fmt$gap$table(shl(density,2)+bytes$per$sector$code);
171   1           read$write$gap=rd$wr$gap$table(shl(density,2)+bytes$per$sector$code);

                  /* initialize system and drivers */
172   1           call initialize$system;
173   1           call initialize$drivers;

                  /* reenable interrupts and give 8272 a chance to report on drive status
                     before proceeding */
174   1           enable;
175   1           call time(10);

                  /* specify disk drive parameters */
176   1           call specify(step$rate,head$load$time,head$unload$time,dma$mode);

177   1           drive=0;                                   /* run single disk drive #0 */

                  /* wait until drive ready */
178   1           do while 1;
179   2             if drive$ready(drive)
                      then go to start;
181   2           end;

182   1           start:
                  call format(drive,density,interleave);
                  |
183   1           do while 1;
184   2             do cylinder=0 to tracks$per$disk-1;
185   3               call seek(drive,cylinder,head);
186   3               do sector=1 to sectors$per$track;

                      /* set up write buffer */
187   4                 do index=0 to bytes$per$sector-1; wrbuf(index)=index+sector+cylinder; end;
```

207885-19

```
 190   4              call write(drive,cylinder,head,sector,density);
 191   4              call read(drive,cylinder,head,sector,density);

                      /* check read buffer against write buffer */
 192   4              if cmpw(@wrbuf,@rdbuf,shr(bytes$per$sector,1)) <> 0FFFFH
                        then halt;
 194   4            end;
 195   3          end;
 196   2        end;

 197   1      end run72;

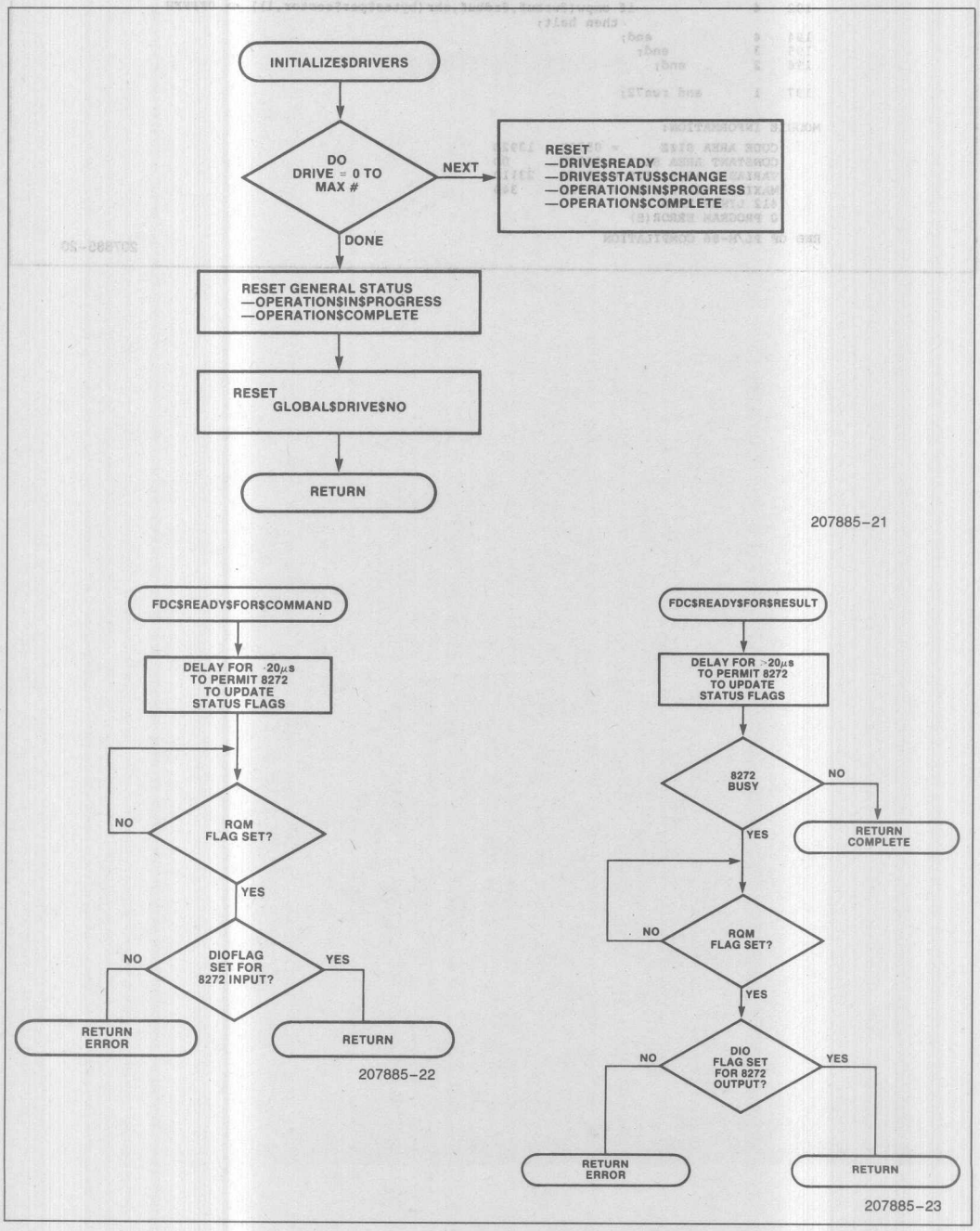
MODULE INFORMATION:

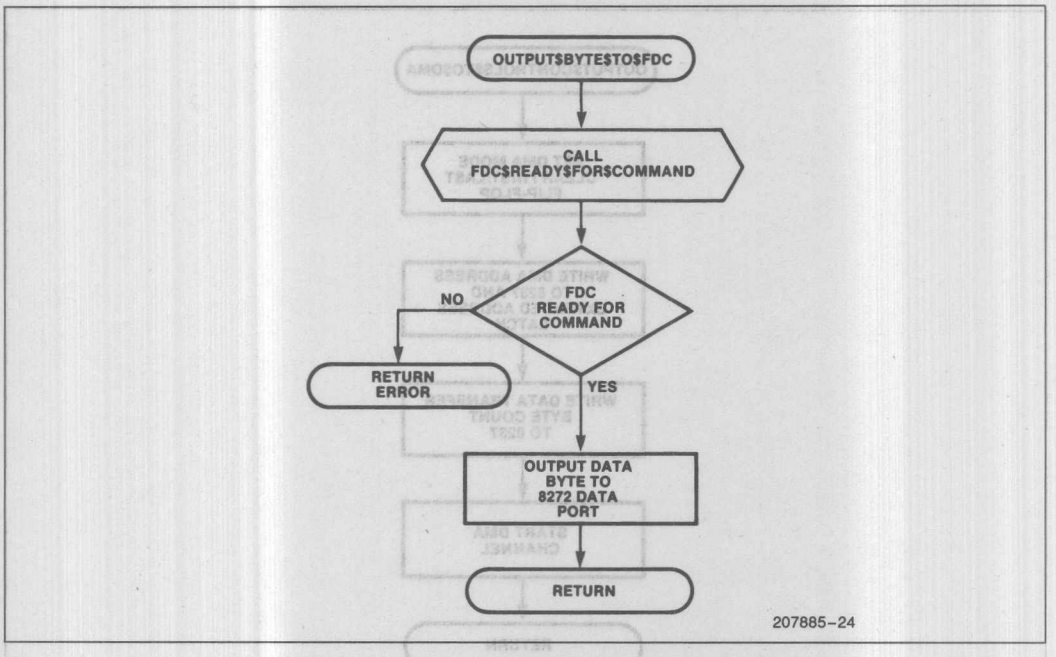
     CODE AREA SIZE     = 0570H   1392D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0907H   2311D
     MAXIMUM STACK SIZE = 0022H     34D
     412 LINES READ
     0 PROGRAM ERROR(S)

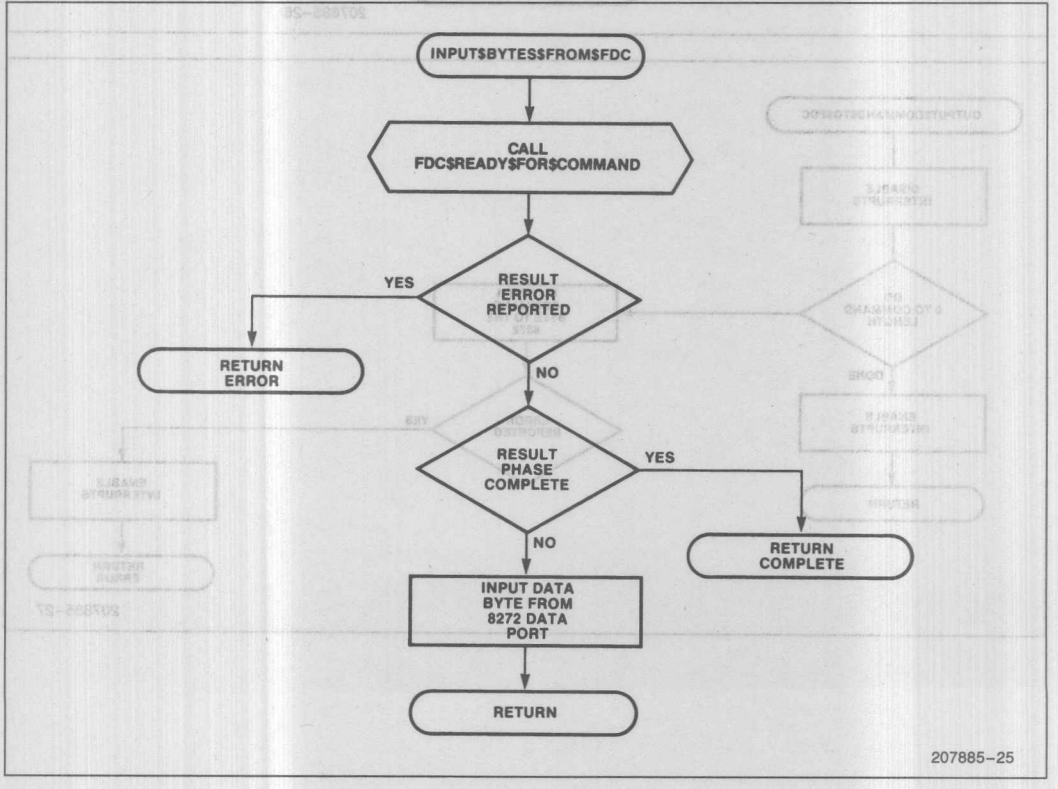
END OF PL/M-86 COMPILATION
```

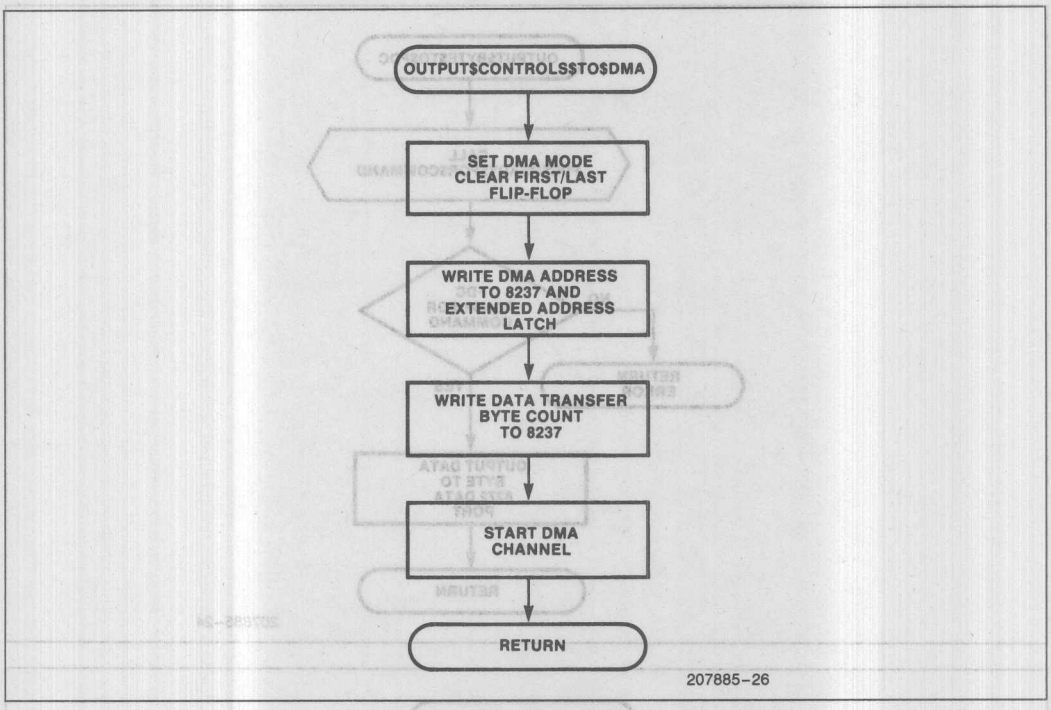
207885-20

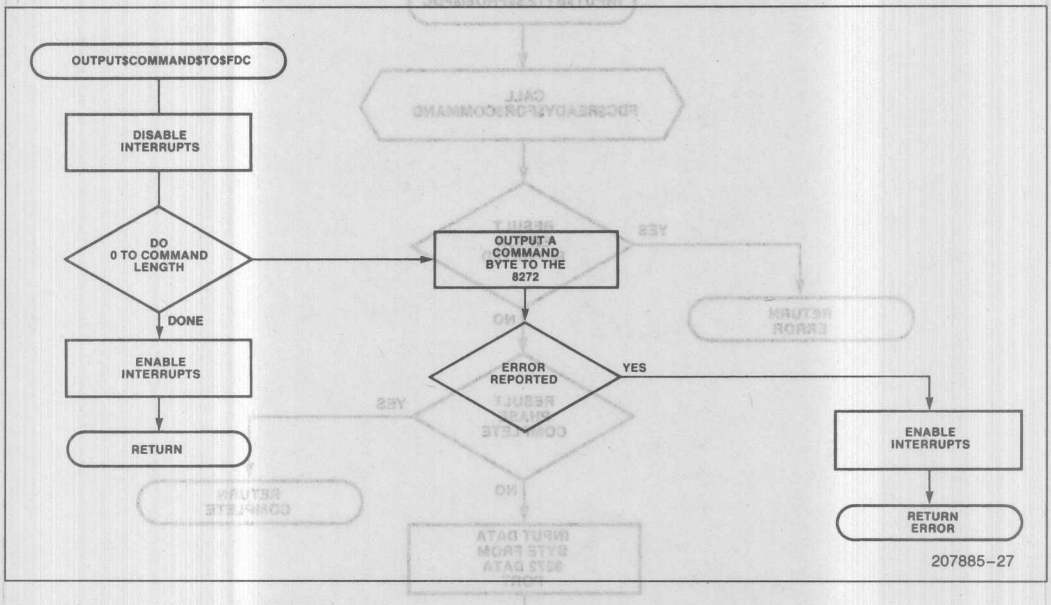
APPENDIX C
8272 DRIVER FLOWCHARTS

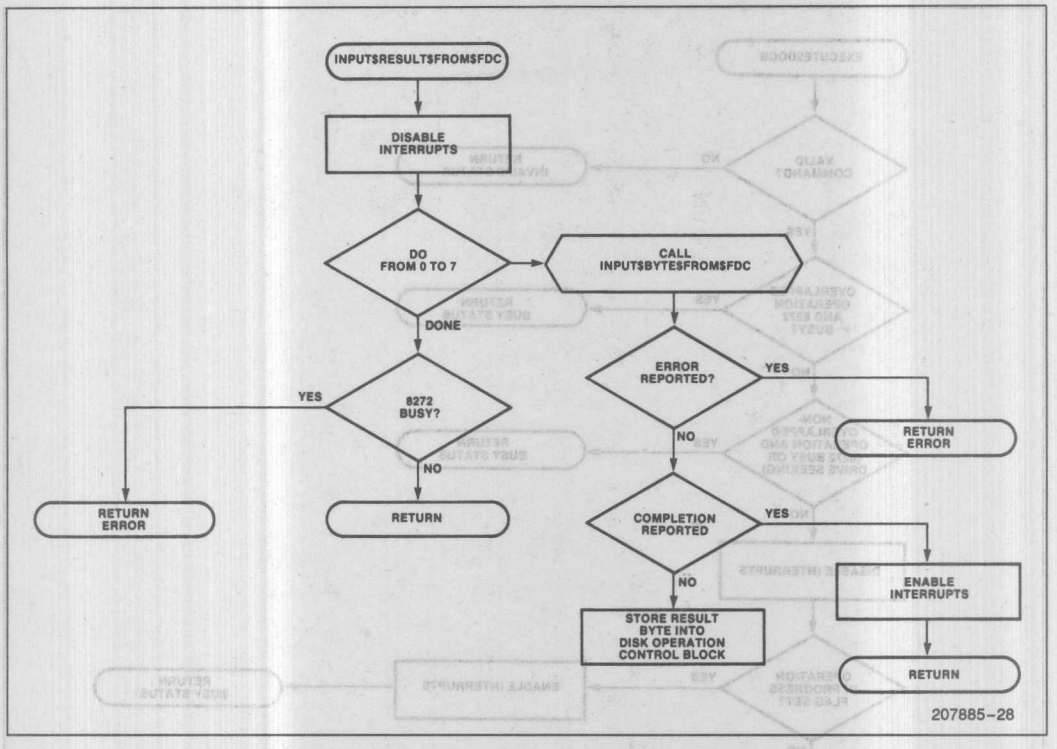



OUTPUT$BYTE$TO$FDC
CALL FDC$READY$FOR$COMMAND
FDC READY FOR COMMAND
NO
YES
RETURN ERROR
OUTPUT DATA BYTE TO 8272 DATA PORT
RETURN
207885-24

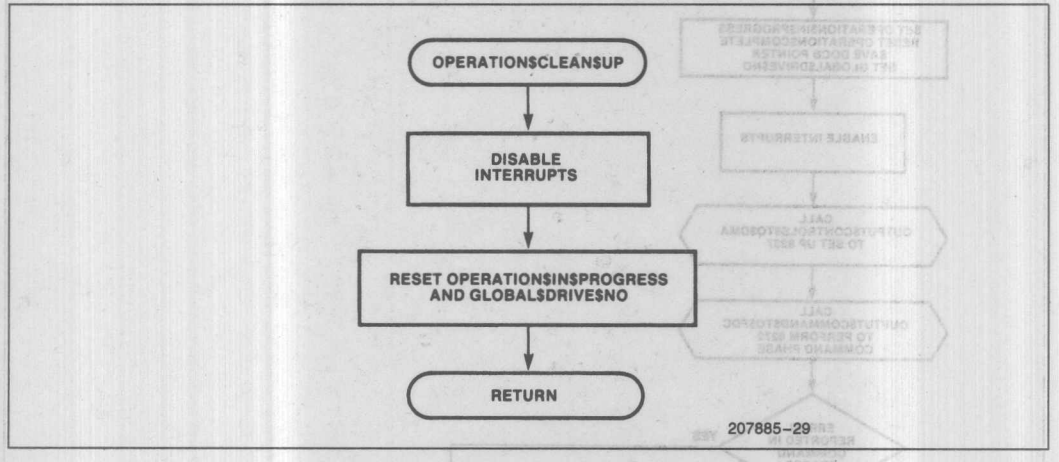

INPUT$BYTES$FROM$FDC
CALL FDC$READY$FOR$COMMAND
RESULT ERROR REPORTED
YES
NO
RETURN ERROR
RESULT PHASE COMPLETE
YES
NO
RETURN COMPLETE
INPUT DATA BYTE FROM 8272 DATA PORT
RETURN
207885-25


OUTPUT$CONTROLS$TO$DMA
SET DMA MODE
CLEAR FIRST/LAST
FLIP-FLOP
WRITE DMA ADDRESS
TO 8237 AND
EXTENDED ADDRESS
LATCH
WRITE DATA TRANSFER
BYTE COUNT
TO 8237
START DMA
CHANNEL
RETURN
207885-26

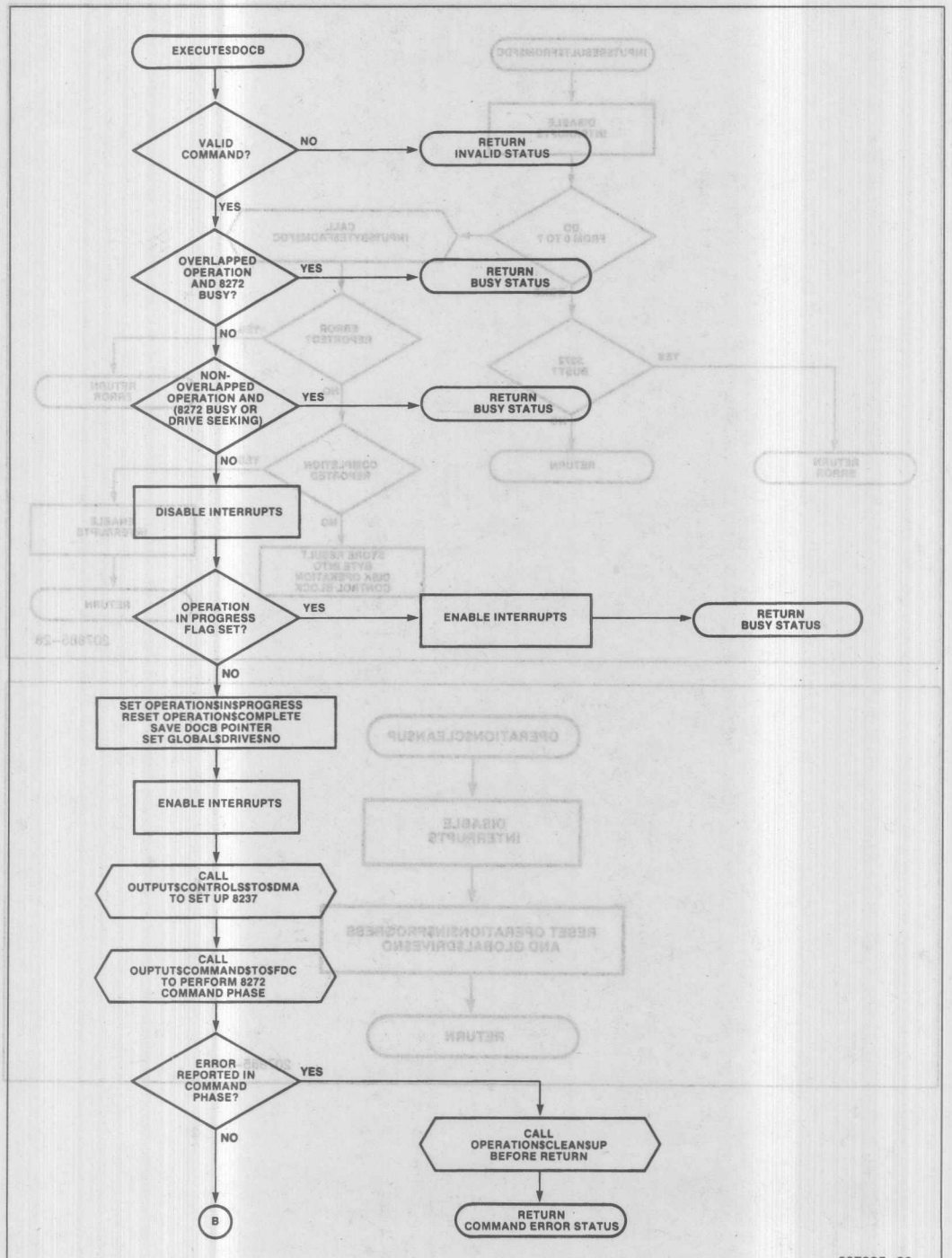

OUTPUT$COMMAND$TO$FDC
DISABLE
INTERRUPTS
DO
0 TO COMMAND
LENGTH
DONE
ENABLE
INTERRUPTS
RETURN
OUTPUT A
COMMAND
BYTE TO THE
8272
ERROR
REPORTED
YES
ENABLE
INTERRUPTS
RETURN
ERROR
207885-27

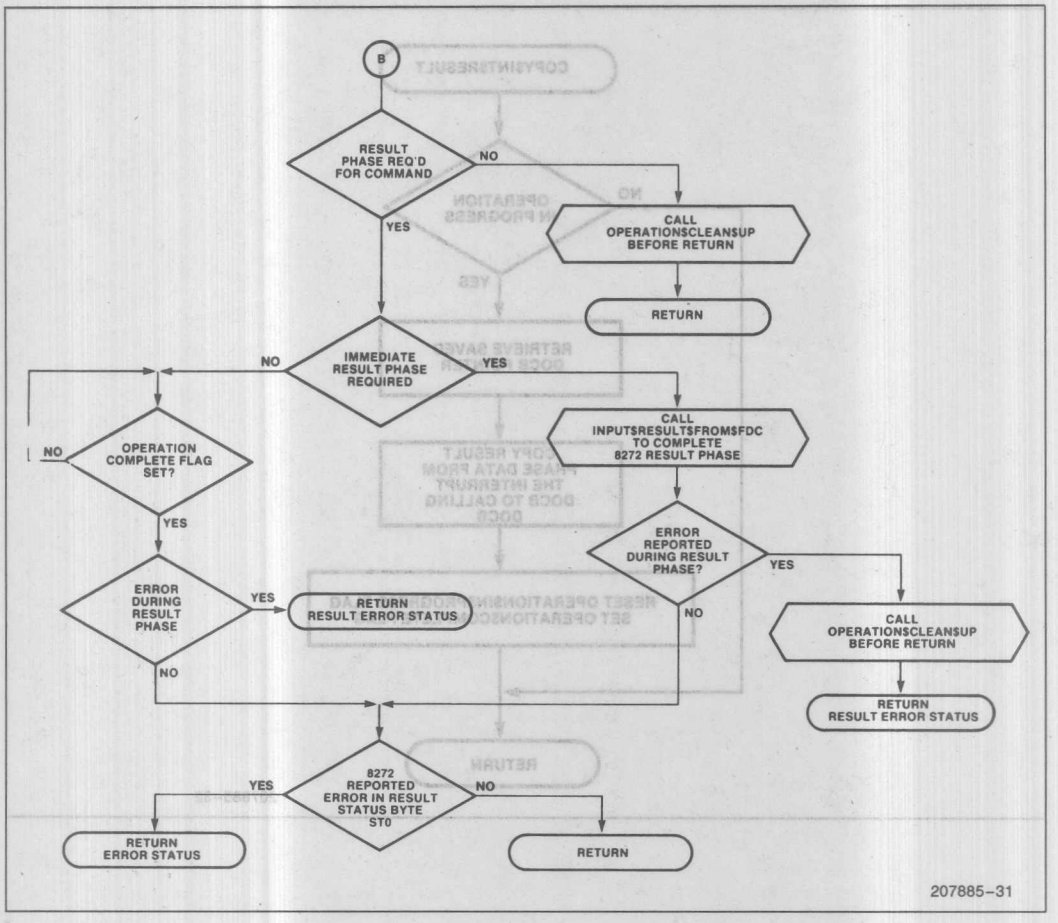


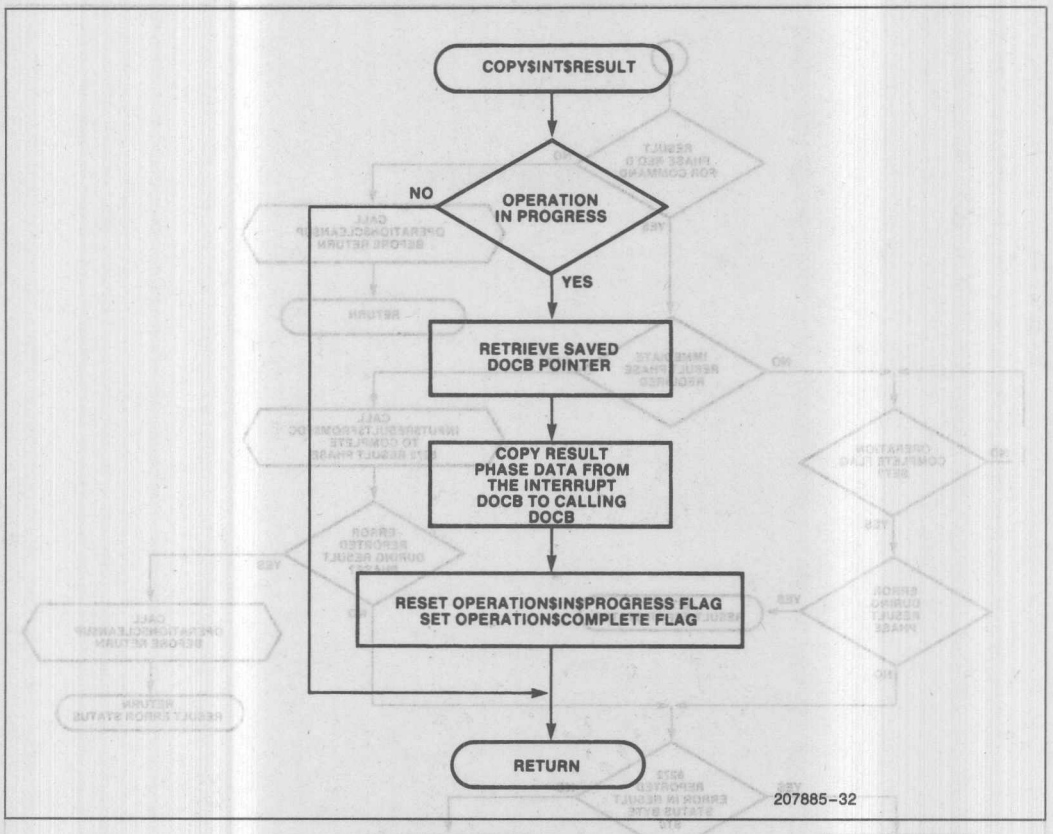
207885-28

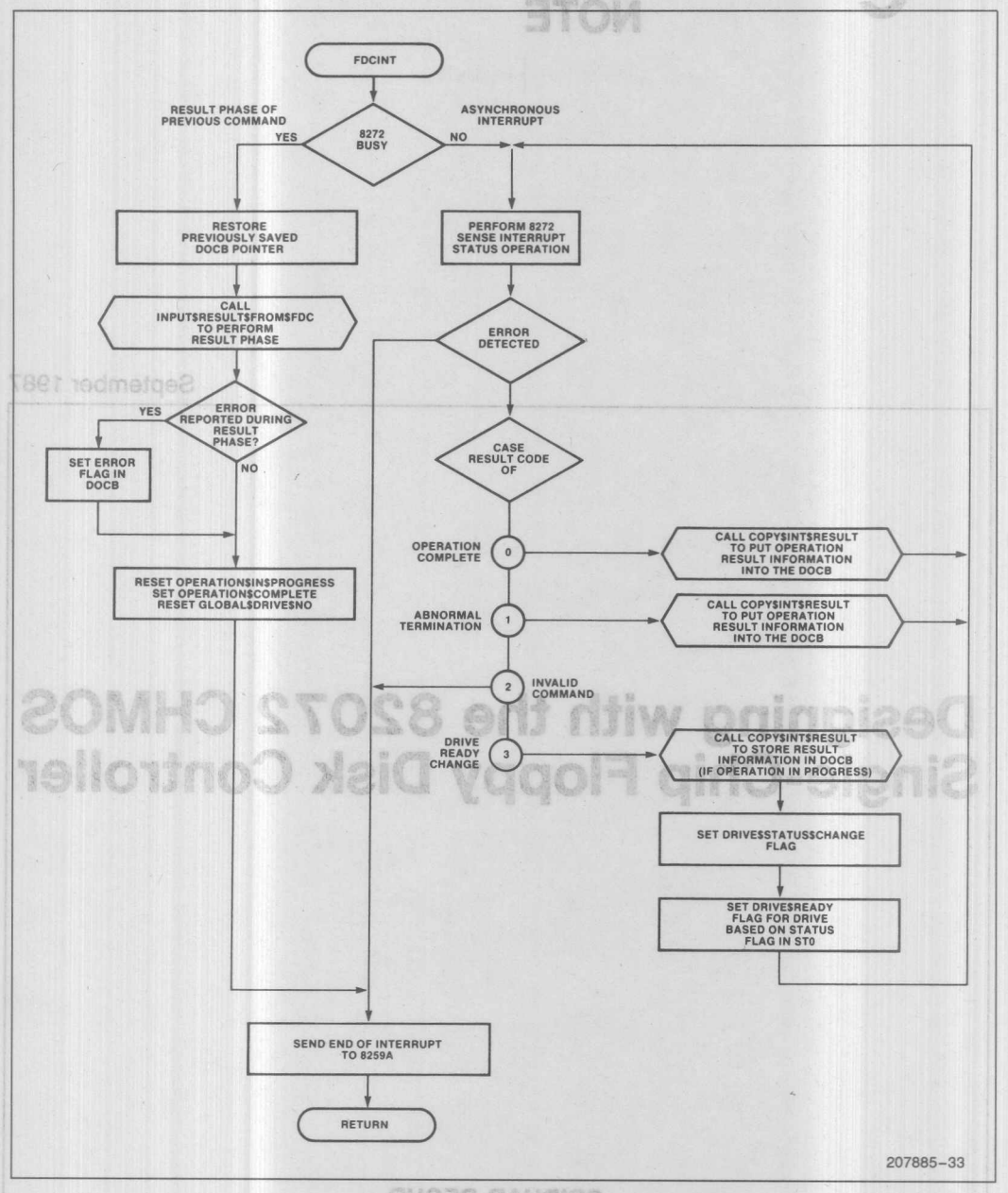


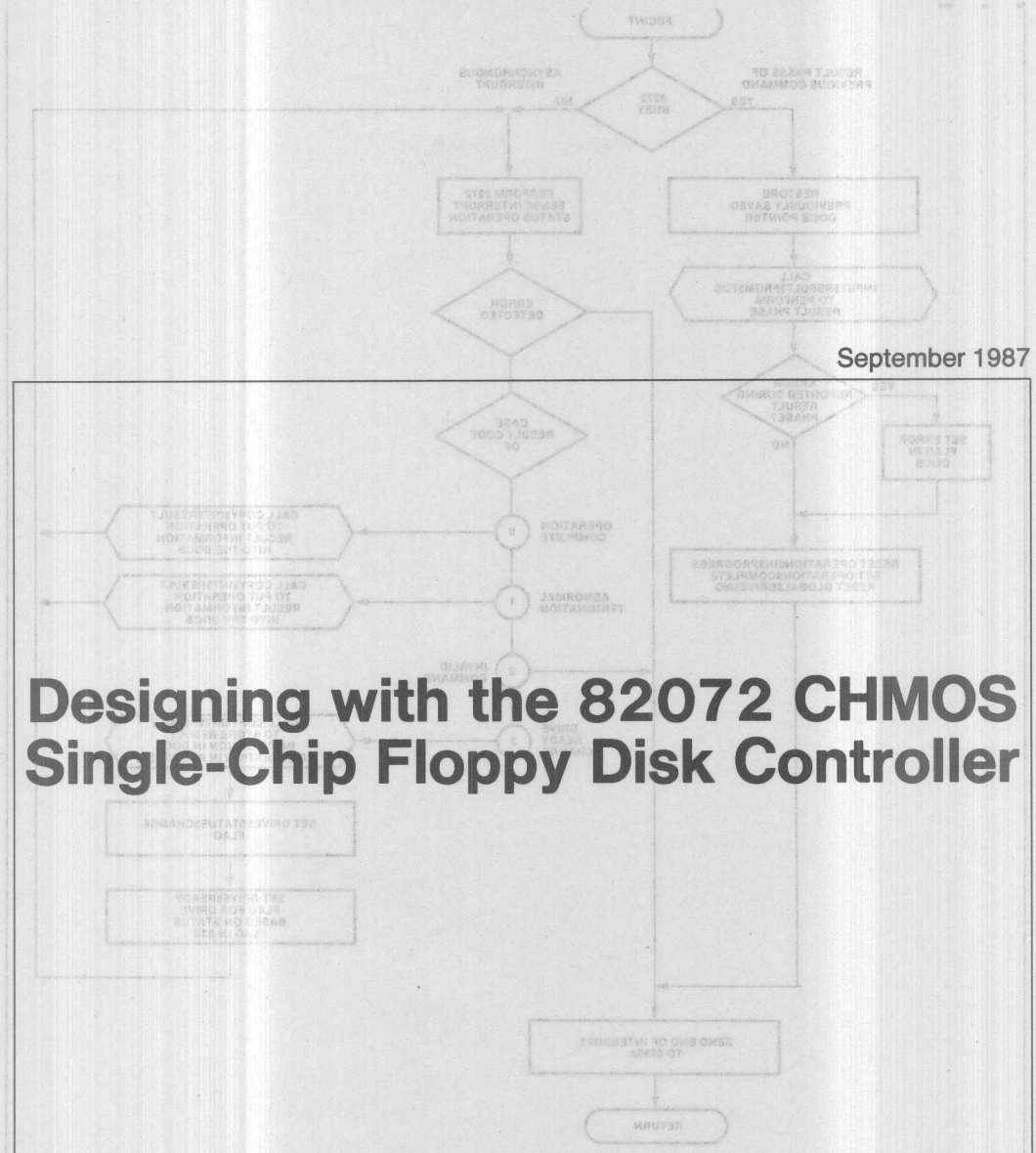
207885-29


EXECUTESDOCB
VALID COMMAND?
NO
RETURN INVALID STATUS
YES
OVERLAPPED OPERATION AND 8272 BUSY?
YES
RETURN BUSY STATUS
NO
NON-OVERLAPPED OPERATION AND (8272 BUSY OR DRIVE SEEKING)
YES
RETURN BUSY STATUS
NO
DISABLE INTERRUPTS
OPERATION IN PROGRESS FLAG SET?
YES
ENABLE INTERRUPTS
RETURN BUSY STATUS
NO
SET OPERATIONINPROGRESS
RESET OPERATION$COMPLETE
SAVE DOCB POINTER
SET GLOBAL$DRIVE$NO
ENABLE INTERRUPTS
CALL OUTPUT$CONTROLS$TO$DMA TO SET UP 8237
CALL OUPTUT$COMMAND$TO$FDC TO PERFORM 8272 COMMAND PHASE
ERROR REPORTED IN COMMAND PHASE?
YES
NO
CALL OPERATION$CLEAN$UP BEFORE RETURN
RETURN COMMAND ERROR STATUS
B
207885-30


B
RESULT PHASE REQ'D FOR COMMAND
NO
YES
CALL OPERATIONSCLEANSUP BEFORE RETURN
RETURN
IMMEDIATE RESULT PHASE REQUIRED
NO
YES
CALL INPUTSRESULTSFROMSFDC TO COMPLETE 8272 RESULT PHASE
OPERATION COMPLETE FLAG SET?
NO
YES
ERROR REPORTED DURING RESULT PHASE?
YES
NO
ERROR DURING RESULT PHASE
YES
RETURN RESULT ERROR STATUS
NO
CALL OPERATIONSCLEANSUP BEFORE RETURN
RETURN RESULT ERROR STATUS
8272 REPORTED ERROR IN RESULT STATUS BYTE ST0
YES
NO
RETURN ERROR STATUS
RETURN
207885-31


COPYINTRESULT
OPERATION IN PROGRESS
NO
YES
RETRIEVE SAVED DOCB POINTER
COPY RESULT PHASE DATA FROM THE INTERRUPT DOCB TO CALLING DOCB
RESET OPERATIONINPROGRESS FLAG
SET OPERATION$COMPLETE FLAG
RETURN
207885-32


FDCINT
RESULT PHASE OF PREVIOUS COMMAND
YES
8272 BUSY
NO
ASYNCHRONOUS INTERRUPT
RESTORE PREVIOUSLY SAVED DOCB POINTER
PERFORM 8272 SENSE INTERRUPT STATUS OPERATION
CALL INPUT$RESULT$FROM$FDC TO PERFORM RESULT PHASE
ERROR DETECTED
YES
ERROR REPORTED DURING RESULT PHASE?
NO
SET ERROR FLAG IN DOCB
CASE RESULT CODE OF
OPERATION COMPLETE
0
CALL COPYINTRESULT TO PUT OPERATION RESULT INFORMATION INTO THE DOCB
ABNORMAL TERMINATION
1
CALL COPYINTRESULT TO PUT OPERATION RESULT INFORMATION INTO THE DOCB
RESET OPERATIONINPROGRESS
SET OPERATION$COMPLETE
RESET GLOBALS$DRIVE$NO
2
INVALID COMMAND
DRIVE READY CHANGE
3
CALL COPYINTRESULT TO STORE RESULT INFORMATION IN DOCB (IF OPERATION IN PROGRESS)
SET DRIVE$STATUS$CHANGE FLAG
SET DRIVE$READY FLAG FOR DRIVE BASED ON STATUS FLAG IN ST0
SEND END OF INTERRUPT TO 8259A
RETURN
207885-33



September 1987

Designing with the 82072 CHMOS Single-Chip Floppy Disk Controller

SRIDHAR BEGUR
APPLICATIONS ENGINEER

Order Number: 292022-003

CHAPTER 1
INTRODUCTION TO THE 82072 FLOPPY DISK CONTROLLER

1.1 ORGANIZATION

This Application Note provides a detailed description of the features and capabilities of the 82072 Floppy Disk Controller. Chapter 1 gives a brief overview of floppy disk concepts. The second chapter "INTERNAL ARCHITECTURE" describes the internal functions of the 82072 in block diagram form. Operation of the internal Data Separator and the Write Pre-Compensation circuits are also included. Chapter 3 presents a functional description of the 82072. Chapter 4 describes the software requirements of the 82072. Chapter 5 describes the technique of interfacing floppy disk drives to the PC/PC-XT, or the PC-AT, using the 82072 floppy disk controller. Appendix A provides detailed flow charts for the various commands, to aid software designers.

Pin functions, electrical and timing characteristics are contained in the 82072 Data Sheet.

1.2 THE 82072 FLOPPY DISK CONTROLLER

The Intel CHMOS 82072 is a fully integrated floppy disk controller designed for use in real time, on-line storage applications. The 82072 has been developed to allow designers to take full advantage of the emerging high capacity, quad density, double sided diskette drive capabilities. The 82072 performs all functions associated with data transfers between disks and user's memory.

The 82072 is a software compatible upgrade of its predecessor, the 8272A floppy disk controller. The on-chip enhancements include a self-compensating analog data separator with software selectable data rates (that require no external components), programmable write pre-compensation circuitry and write clock generation logic. These enhancements greatly reduce the number of components required to interface high capacity floppy disk drvies to processors. The on-chip sixteen byte FIFO, with programmable threshold, minimizes data transfer times and overcomes the I/O bottleneck for fast disk access. Additionally, the 82072 has a power down mode which makes it ideal for low power, battery backed, portable applications.

1.3 OVERVIEW

The following sections provide the reader with a simplified overview of floppy disk controller concepts. These sections are particularly appropriate for those who are unfamiliar with floppy disk operations.

1.3.1 Data Separator

The actual signal recorded on a floppy disk is a combination of timing information and data. The serial read data from the disk must be converted into two signal streams: clock and data. The serial data must also be assembled into eight bit bytes for transfer to the host. The Data Separator generates a Data Window signal that is active during the data bit cell interval. This Data Window signal is used by the READ hardware to extract data from the serial read data stream.

The job of data separation is relatively straightforward for the single density encoding scheme. In this scheme, each bit cell contains a clock bit at the leading edge of the cell. The bit cell is 4 microseconds wide (at 125 Kbps data transfer rate). The data bit (if present) is always located at the center of the cell. The job of the data separator is to generate a data window 2 microseconds wide starting one microsecond after the clock pulse. Since every cell has a clock bit, a fixed window reference is available for every bit, and as the window is 2 microsecond wide, a slightly shifted data bit will still remain within the data window.

When MFM (Modified Frequency Modulation) encoding is used, data separation has two problems. First, MFM encoding loses the fixed cell reference pusle present in FM encoding as only some bit cells contain a clock bit. Secondly, the bit cell in MFM is one half the size of the bit cell for FM. This shorter bit cell implies that MFM cannot tolerate as large a playback data shift (as FM can) without errors.

The data separator for MFM must continuously monitor the read data stream, synchronizing its operation with the actual clock/data bits of the data stream. The data separation circuit must track the input Read Data very closely as unpredictable bit shifts leave less than 50 ns of margin with respect to Data Window to decode data reliably.

1.3.1.2 PHASE LOCK LOOP (PLL)

Only an analog PLL can provide the reliability required for double density data separation. A PLL is an electronic circuit that continuously analyzes the input frequency and locks another reference oscillator to that frequency. The PLL determines the clock and data bit position by sampling each bit in the serial data stream. The phase difference between the PLL generated Data Window and the data bit is constantly fed back to adjust the position of the data window, enabling the PLL to track input data frequency changes. This allows the data separator to reliably read back previously recorded data.

The 82072 has an on-chip self-compensating analog data separator. The data separator is a fully integrated design with all passive and active components on chip.

The internal data separator consists of two analog Phase Lock Loops (PLL): a Reference PLL and a Data PLL. Analog Phase Locked Loops were chosen to provide the reliability and resolution required for double density data separation. The Reference PLL stabilizes the Data PLL. The Data PLL performs the actual data separation function by synchronizing its operation to the digital read data input from the disk drive. The synchronized signal (Data Window), generated by the Data PLL, is used to separate the clock and data bits from the read data signal. The data bits are then assembled into bytes and transferred to the system memory.

The PLL consists of three main components: a phase comparator, a second order loop filter and a Voltage Controlled Oscillator. The block diagram of the PLL is shown in Figure 1.0. Basically, the PLL compares the phase difference between the VCO output and the read data input from the disk drive. The difference in phase is used to increase or decrease the frequency of the voltage controlled oscillator in order to bring its frequency closer to that of the input. The PLL is locked when the frequency of the oscillator is exactly the same as the frequency of the read data input.

1.3.2 Data Rate

The 8272A allows the user to vary the data transfer rate by changing the frequency of the write clock. To maintain this flexibility, the 82072 provides an on-chip prescaler that divides the 24 MHz clock to rates that are equivalent to those used with the 8272A. By appropriately setting the DRATSEL bits in the Data Rate Select (DSR) register, the prescaler value can be changed to obtain the desired data transfer rate. The supported data transfer rates are 250, 300, 500 Kbps and 1 Mbps with MFM encoding, and 125, 150 and 250 Kbps with FM encoding. Most of the internal circuitry of the 82072 is driven with a 2-phase clock (PH1, PH2) that is synchronous with the chosen data rate.

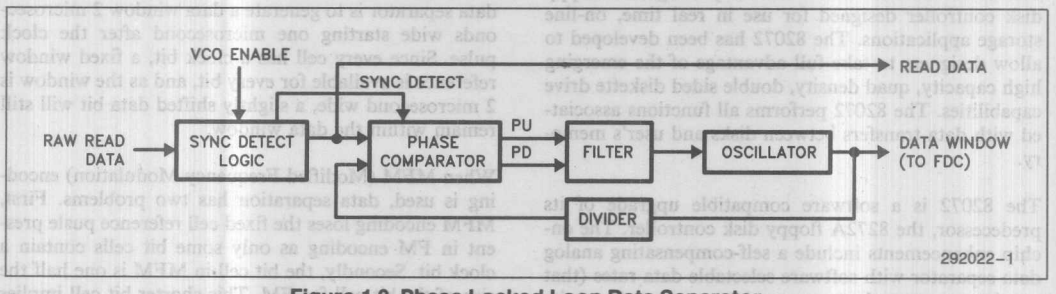


Figure 1.0. Phase Locked Loop Data Separator

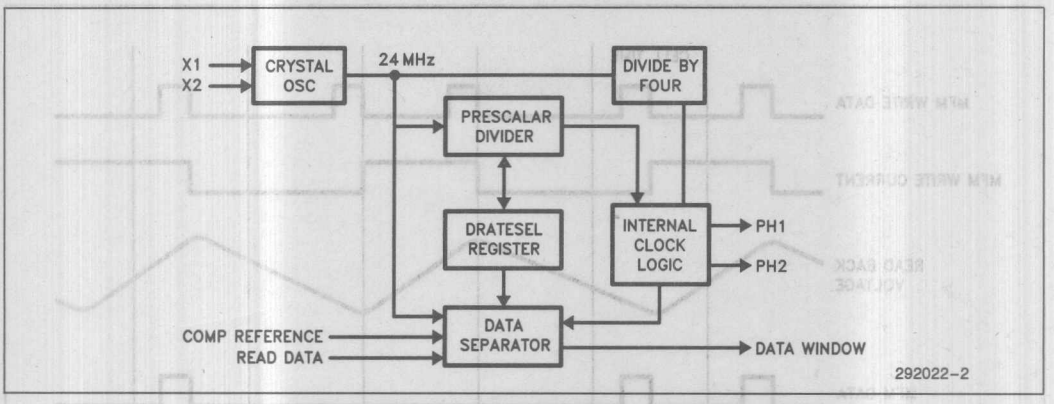


Figure 1.1. 82072 Clock Generation Logic

Table 1.1. Supported Data Transfer Rate

DRATESEL		PH1/PH2	WRCLK	
Bit 1	Bit 0		MFM	FM
1	1	8.0 MHz	1 MHz	—
0	0	4.0 MHz	500 Kbps	250 Kbps
0	1	2.4 MHz	300 Kbps	150 Kbps
1	0	2.0 MHz	250 Kbps	125 Kbps

1.3.3 Write Pre-Compensation

Data is recorded on the magnetic media by pulsing a recording head to produce a flux change on the surface of the diskette for each logic one. Lack of such a magnetic pulse indicates a logic zero. Ideally, the flux reversal induced by the Read/Write head would be instantaneous, as shown in Figure 1.1. Current would immediately switch from one polarity to the other. But in reality, it takes time for the current to reverse and the fields to decay and build up in the opposite direction. The resulting read back voltage is more or less sinusoidal with peaks less defined in time or amplitude.

With current recording techniques, adjacent clock/data pulses are close enough to interact with each other. This phenomenon is particularly noticeable on the inner tracks. Peak shift is the result of the interaction of these pulses. Because the two pulses tend to have a portion of their individual signals superimposed on each other, the actual read back voltage is an algebraic summation of the pulses, giving rise to overlap errors.

When all "1's" or all "0's" are being recorded, the data frequency remains constant and the pulses are uniformly spaced. As a result, the positive going and the negative going errors tend to be equal and opposite and hence cancel each other. This is a "zero peak shift" condition.

Peak shift occurs when there is a change in frequency. A "011" pattern represents a frequency increase since there is a delay of about 1.5 cells between the "01" and only 1 cell spacing between the "11". As a result, the squeezing of cells cause the mathematical average (the actual read back voltage) to shift the apparent peak to the left. This is known as "early peak" shift.

A "110" pattern represents a frequency decrease since a pulse is not written at all in the third cell. This causes the peak to be shifted to the right. This is known as "late peak shift."

The effect of peak shift is to displace the flux changes and create a timing uncertainty which can cause errors when data is read back. MFM encoding is much more susceptible to peak shift errors due to the shorter data window. The data window is defined as the time that is allowed for the data bit to appear and be recognized. The data window of the MFM encoding is 1 microsecond (at 500 Kbps) as opposed to the data window of FM encoding where the Data Window is 2 microseconds. The MFM encoding cannot tolerate as large a play-back data shift (as compared to FM encoding) without errors, due to the smaller data window.

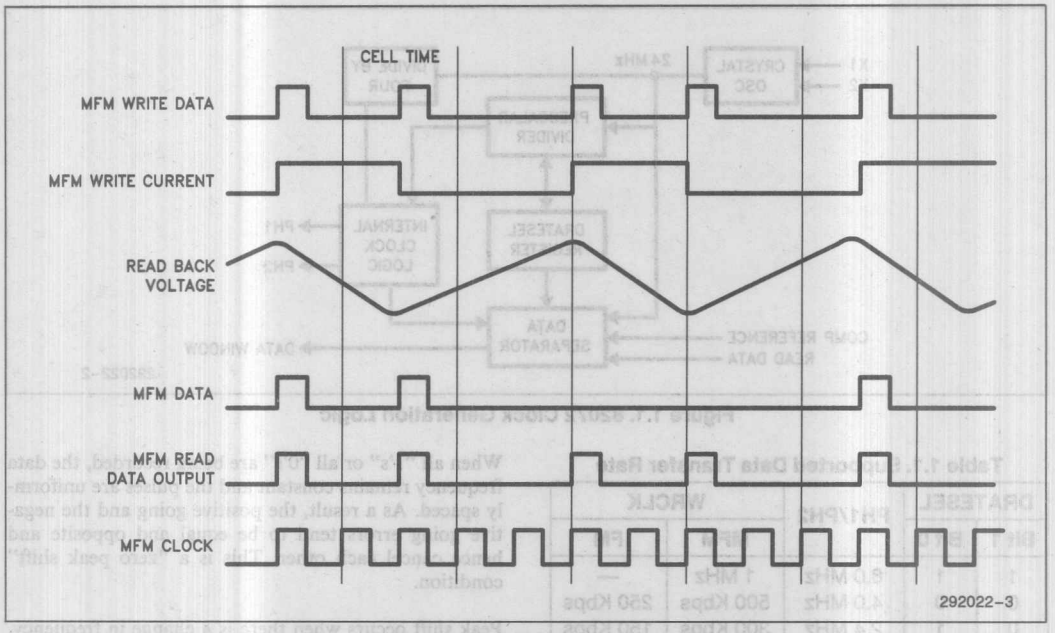


Figure 1.2. MFM Recording

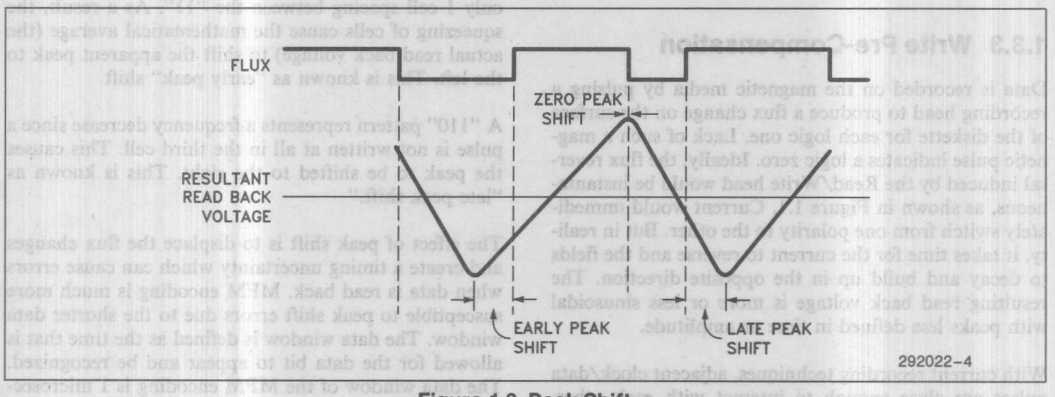


Figure 1.3. Peak Shift

To overcome this problem, the WRITE DATA stream is pre-compensated, i.e., the direction of each bit shift is anticipated and the bit is moved in the opposite direction before being written.

The 82072 has on-chip write pre-compensation circuitry. This circuitry pre-compensates for head and media caused peak shifts. The amount of pre-compensation applied is programmable, and the value should be chosen to compensate for the media peak shifts.

The older 8″ drives required a pre-compensation value of 250 ns. The newer 5¼″ and 3½″ mini floppy disk drives require a pre-compensation value of 125 ns. The 82072 supports both pre-compensation values.

In order to support tomorrow's higher capacity disk drives (1 Mbps data transfer rate), the 82072 allows the user to select lower pre-compensation values. The pre-compensation values supported by the 82072 are summarized in Table 1.2.

Table 1.2. Write Pre-Compensation Values

Pre-Comp	Amount of Pre-Comp	Frequency
111	0 ns—Disabled	0
001	41.67 ns	1/24 MHz
010	83.34 ns	2/24 MHz
011	125.00 ns	3/24 MHz
100	166.67 ns	4.24 MHz
101	208.33 ns	5.24 MHz
110	250.00 ns	6.24 MHz
000	Default	

If the 82072 is programmed with the default precompensation value (000), a default pre-compensation value matching the PC-AT is automatically chosen based on the data rate selected. The default values corresponding to the data rate values are illustrated below in Table 1.2.1.

Table 1.2.1. Default Write Pre-Compensation Delays

Data Rate	Pre-Compensation Delay
1 Mbps	41.67 ns
500 Kbps	125 ns
300 Kbps	125 ns
250 Kbps	125 ns

1.3.4 Seek

There are three ways to position the disk Read/Write head over the desired cylinder:

1. Normal Seek
2. Implied Seek
3. Relative Seek

1. Normal Seek

This uses the SEEK command. During the execution phase of this command, the track number to seek to (NCN) is compared with the present track number (PCN). If there is a difference, the following operation is performed:

1. PCN < NCN: The DIRECTION signal to the disk drive is set to "1" (step-in) and step pulses are issued to position the head over the specified cylinder.
2. PCN ≥ NCN: The DIRECTION signal is set to "0" (step-out) and the step pulses are issued to retract the read/write head to the cylinder specified.

2. Implied Seek

This method uses the implied seek mode. When enabled, the 82072 automatically performs a seek operation before executing a read or write command.

With the 8272A, a READ operation required the following steps:

1. Issue SEEK command
2. Issue Sense Interrupt Status Command
3. Issue Read Data Command.

The new "IMPLIED SEEK" feature is enabled by issuing a "CONFIGURE" command. When enabled, only the data transfer command needs to be issued. The 82072 automatically performs the SEEK and the Sense Interrupt Status commands before executing the data transfer command.

3. Relative Seek

Standard Double Density diskettes (360 Kbytes) have 48 tracks per inch (tpi) while high capacity diskettes (1.2 Mbytes) have 96 tracks per inch. The emerging high capacity floppy disk drives can support up to 10 Mbytes and have more than 300 tracks.

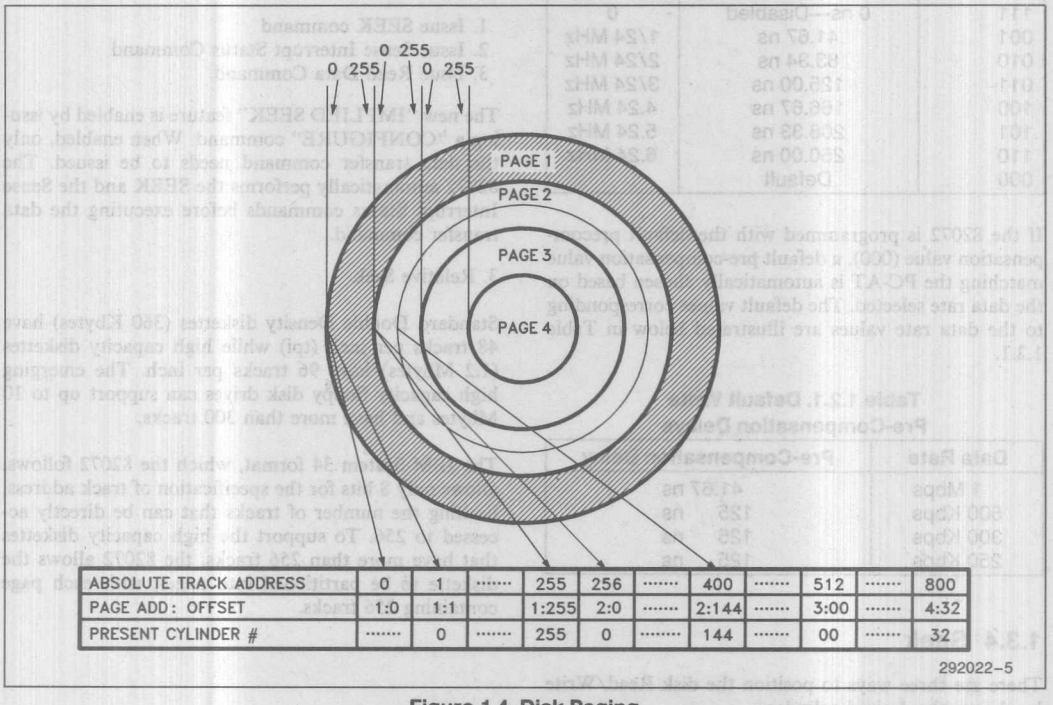
The IBM System 34 format, which the 82072 follows, allows only 8 bits for the specification of track address, limiting the number of tracks that can be directly accessed to 256. To support the high capacity diskettes that have more than 256 tracks, the 82072 allows the diskette to be partitioned into pages, with each page containing 256 tracks.

To support this paging mechanism, the 82072 provides a "Relative Seek" mode. The Relative Seek command differs from the Seek Track command in that it steps the head the absolute number of tracks specified in the command, instead of making a comparison against an internal register.

The paging mechanism can be better understood by considering an example. Let us assume that a diskette has 1024 tracks. This would be divided into four pages with each page containing 256 tracks. Let us also assume that the CPU needs to access track 800 (which is contained in Page 4). If the Read/Write head is currently on track 400 (contained in Page 2; the PCN register contains 400 − 256 = 144), then a total of 800 − 400 = 400) 400 step pulses have to be issued, to position the head over the desired track. The maximum number of step pulses that can be issued by a single seek command is 256 pulses (as the relative cylinder number register is 8 bits wide). The internal register, PCN (which is an 8-bit register that stores the track that the R/W head is currently on), will overflow as the cylinder number crosses the track 255 and will count up from 0.

The approach would be to issue two successive Relative seek commands. The first Relative Seek is issued with the Relative Seek Number (RCN) register initialized to 255 (which will result in 256 step pulses). When this command is executed, the Read/Write head will be positioned over the absolute track 656 (400 + 256) (absolute track address). The PCN register will now contain (144 + 256 − 256) 144. The second relative seek is issued with the RCN register initialized to 143 (which will result in 144 Step pulses). This will position the Read/Write head over track 800 and the PCN register will contain (144 + 144 − 256 = 32) 32.

When operating within a page all commands function correctly. The user's software should keep a record of the Page that the Read/Write head is currently on, and determine the number of times the Relative Seek command is executed, to position the Read/Write head over the desired cylinder.



ABSOLUTE TRACK ADDRESS	0	1	·····	255	256	·····	400	·····	512	·····	800
PAGE ADD: OFFSET	1:0	1:1		1:255	2:0	·····	2:144	·····	3:00	·····	4:32
PRESENT CYLINDER #	·····	0	·····	255	0	·····	144	·····	00	·····	32

Figure 1.4. Disk Paging

1.3.5 FIFO

The microprocessor interface was enhanced by adding a 16 byte FIFO to reduce the timing constraints that most floppy disk controllers impose upon a system. The 16 Byte FIFO with programmable threshold eliminates the extremely critical timing constraints by permitting burst data transfers between the 82072 and the system bus. The point at which the 82072 generates a request for data transfers is selectable within the 16 byte range. The interface was further enhanced to support today's faster microprocessors (i.e., 10 MHz 80186, 10 MHz 80286) without incurring waitstates.

1.3.6 Power Down Mode

The 82072 power down mode allows the internal oscillator to be turned off by the user's software. Because the 82072 is fabricated in CHMOS, the current consumption occurs only when the transistors change state. In power down, the controller consumes less than 125 microamps.

Table 1.3. 82072 Feature Summary

Features	PC-AT Implementation	FDC 82072
Write Pre-Comp	External/Programmable	On-Chip/Programmable
Write Clock	External/Programmable	On-Chip/Programmable
Drive Select	External	On-Chip
Motor On	External	On-Chip
Data Separator	External/Programmable	On-Chip/Programmable
Software Reset	External/Software Contr	On-Chip/Software Contr
Power Down	Not Supported	Supported/Software Contr
Low Density Signal	External	On-Chip

Table 1.4. 82072 Supported Data Rates

		Values	PC-AT	FDC 82072
Supported Data Rates	MFM	250 Kbps	yes	yes
		360 Kbps	yes	yes
		500 Kbps	yes	yes
		1 Mbps	no	yes
	FM	125 Kbps	—	yes
		150 Kbps	—	yes
		250 Kbps	—	yes
Supported Pre-Comp Values		0 ns	no	yes
		41.67 ns	no	yes
		83.34 ns	no	yes
		125.00 ns	yes	yes
		166.67 ns	no	yes
		208.33 ns	no	yes
		250 ns	no	yes

ARCHITECTURAL OVERVIEW

2.1 ARCHITECTURE

The 82072 provides the data interface between the floppy disk drive and the processor. The 82072 is a software compatible upgrade of the 8272A, with many on-chip enhancements. Figure 2.1 shows the block diagram of the 82072. The highlighted blocks represent areas that are completely new or highly enhanced. All the new features were adopted with the requirement of being software compatible with the 8272A. The new features of the 82072 are:

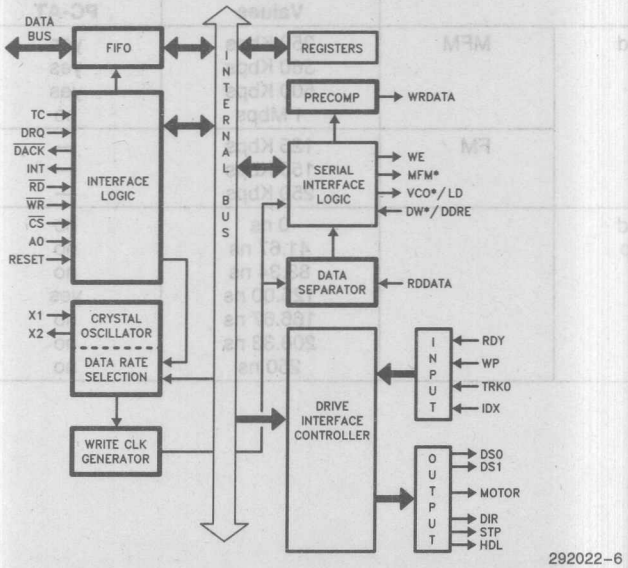
1. A crystal controlled reference oscillator
2. Pre-compensation circuits for the write operation
3. Analog Data Separator for data detection in the read mode
4. Motor enable and drive select logic
5. FIFO (Chapter 3)
6. Enhanced Processor Interface (Chapter 3)

2.1.1 Crystal Controlled Oscillator

The 82072 requires an external 24 MHz clock that can either be supplied by a 24 MHz crystal or a MOS level square wave input. All the internal timings are derived from this clock input. It provides the reference timings for recording of data, generation of Write Clock, pre-compensation circuitry and the analog data separator.

The 82072 provides the Write Clock for synchronization of control signals and Write Data, during the write operation. To retain the flexibility of the 8272A, where the user could vary the data transfer rates by changing the frequency of the Write Clock, a prescalar was added to the Write Clock generation logic, which divides the 24 MHz clock to generate the Write Clock. By appropriately setting the data rate select bits in the DSR the prescalar value can be changed to obtain the desired Write Clock frequency.

The oscillator also serves as a reference to the read circuits. While waiting for the synchronization field, preceding the data field, the PLL is locked to a reference clock frequency (derived from the 24 MHz clock input) that is 2 or 4 times the selected data rate for MFM and FM respectively. This allows a faster aquisition time when a valid data stream arrives.



NOTE:
These pins are not used with the internal PLL.

Figure 2.1. 80272 Block Diagram

To overcome the undesirable effects of peak shifts (the cause and effect of peak shift is described in CHAPTER 1), the 82072 pre-compensates the Write Data stream before sending it to the disk drive.

The amount of pre-compensation applied depends upon the data transfer rate and the characteristics of the recording media and is typically 125 ns (for 500 Kbps). This pre-compensation is applied to data patterns that result in large peak shifts. The block diagram of the pre-compensation logic is shown in Figure 2.2. The Write Data is fed into a 13 bit serial in-parallel out shift register. The shift register is clocked at the main clock rate (24 MHz). The no-delay tap is at the center (bit 7) of the shift register. This allows six levels of early and late shifting (in increments of 41.67 ns) with respect to the no-delay tap. The first six outputs are fed into an Early Multiplexer and the last six outputs are fed into a Late Multiplexer. The multiplexers select one out of the six inputs depending upon the programming of the Data Rate Select register. A final multiplexer combines the early, late and normal signals to generate the Write Data to the disk drive. The final multiplexer is controlled, in turn, by PS0 and PS1 (8272A equivalent signals), which is internally generated by the 82072.

Table 2.1. PS0, PS1 decoding

PS0	PS1	Select
0	0	Normal
0	1	Late
1	0	Early
1	1	Invalid

The circuit examines four bits and determines whether to shift or not. The following patterns are compensated (bit shifted) in the direction of the arrow.

→	←
X011	X110
1000	0001

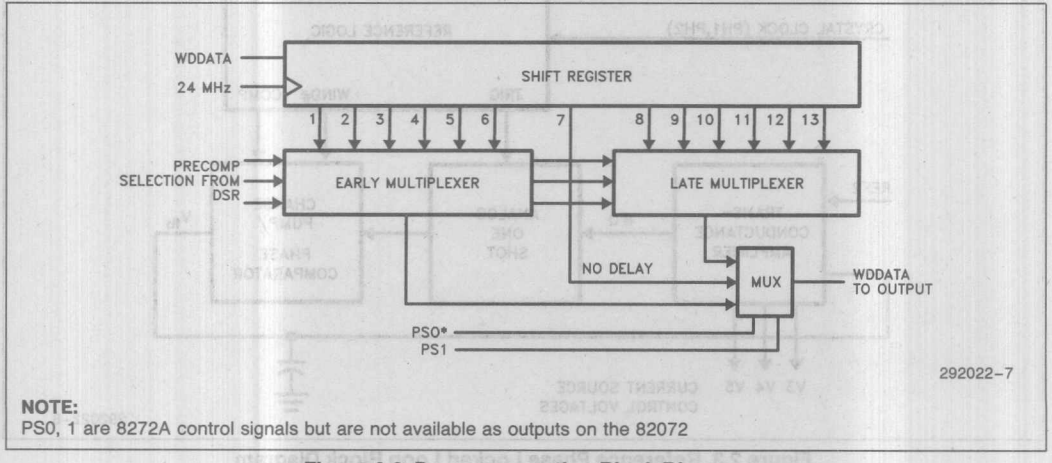
X = don't care.

When a flux transition pattern of "110" is written on to the disk the second "1" is pulled towards the "0". The write pre-compensation logic shifts this bit in the opposite direction by an amount equal to the expected shift.

Since the play-back shift is predictable, pre-compensation can be applied to the write data stream, such that the clock and data bits can be correctly positioned for subsequent reads. This function is completely controlled by the 82072 and is required only for MFM encoding. During write operations, the 82072 specifies an early, late or normal bit positioning depending upon the data pattern. This timing is specified with respect to the 82072 generated write clock. The track on which write pre-compensation is to be enabled is programmable.

2.1.3 82072 Data Separator

The 82072 data separator consists of two analog PLLs: a Data PLL and a Reference PLL. The Data PLL performs the actual data separation. The Reference PLL stabilizes the Data PLL. Unlike conventional PLLs which are based on a Voltage Controlled Oscillator, the Reference PLL uses an analog mono-stable multivibrator. The frequency input to the Reference PLL is a derivative of the 24 MHz crystal oscillator input. In lock, the output pulse width of the mono-stable multivibrator matches the pulse width of the input reference frequency.



NOTE:
PS0, 1 are 8272A control signals but are not available as outputs on the 82072

Figure 2.2. Precompensation Block Diagram

The Reference PLL controls the Data PLL's center frequency, charge pump current and the transfer function of the loop filter.

The Data PLL in conjunction with the Reference PLL provides a Data Separator that is immune to temperature, voltage, process and data rate variations. The analog data separators available today cannot provide this immunity.

2.1.3.1 FUNCTIONAL OVERVIEW

The 82072 uses two signals to select the transfer rate of the Data Separator:

1. MFM/FM sets the coding mode. This bit directly controls the reference frequency of the Data PLL.
2. LD 500 Kbps or 300/250 Kbps which is controlled by the data rate signals. The source of the data rate signals is the DSR which is written to by the user's software.

The reference frequency to the Data loop is controlled by a combination of the Reference PLL's analog information and the Data PLL's response to the reference signal input. Table 2.2 shows the relationship between the data rates and the input reference frequency to the Data PLL.

Table 2.2 Frequency Chart

Data Rate	Ref Freq	VCO Freq	DW Freq MFM	DW Freq FM
1 Mpbs	2 MHz	4 MHz	1 MHz	500 KHz
500 Kbps	1 MHz	2 MHz	500 KHz	250 KHz
300 Kbps	600 KHz	1.2. MHz	300 KHz	150 KHz
250 Kbps	500 KHz	1 MHz	250 KHz	125 KHz

During idle, while the VCO enable signal is inactive, the Data PLL is locked on to the input reference frequency. When the 82072 receives a data transfer command from the host, it internally activates the VCO enable signal (if the internal data separator is being used) and enables the sync detection logic. When a data transfer command is received, the sync detection logic samples the Read Data input for eight consecutive ones. When the eight sync pulses are detected, the Data PLL's input is switched from the reference frequency to the Read Data input. The Data PLL's VCO is halted and restarted in phase with the incoming Read Data input. The Data PLL generates the Data Window signal that is used by the read hardware to separate clock and data information.

2.1.3.2 Reference PLL Description

The primary function of the Reference PLL is to stabilize the Data PLL. The reference PLL is not directly involved in the data separation process.

The block diagram of the Reference PLL is shown in Figure 2.3. The principle elements of the Reference PLL are a phase comparator, charge pump, transconductance amplifier, analog mono-stable multivibrator and a reference signal generator. The pulse width of the mono-stable multivibrator (one shot) is controlled by a current source. The current source is, in turn, controlled by a signal called IFQ which is generated by the

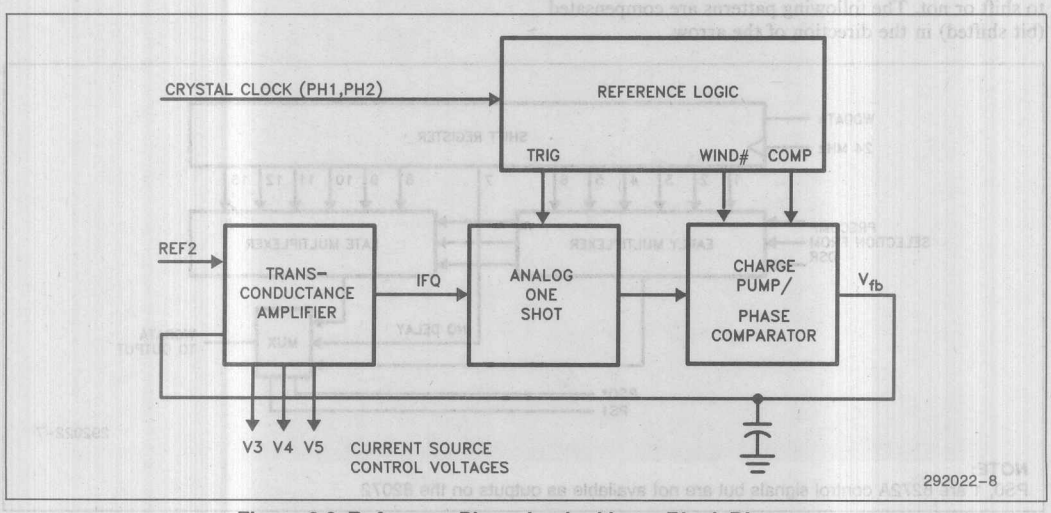


Figure 2.3. Reference Phase Locked Loop Block Diagram

transconductance amplifier. The higher the current, the shorter the pulse width. The transconductance amplifier also generates current source control voltages V3, V4, V5. The V_3, V_4, and V_5 signals control the center frequency of the Data PLL's VCO. The levels of these signals are dependent on Vfb (the Reference PLL's feed-back voltage-Vfb). As Vfb increases the current source's current increase. The transconductance amplifier provides a voltage to current conversion with Vfb being the reference input voltage.

As mentioned earlier, the reference clock ($\overline{WIND}$) is a derivative of the 24 MHz crystal clock input. The circuitry that generates the reference clock provides two additional signals—TRIG and $\overline{COMP}$. TRIG triggers the one shot every time the reference clock makes a low to high transition. When TRIG is active, the $\overline{COMP}$ signal disables the phase comparator of the Reference PLL.

The Reference PLL is locked onto the stable reference clock (for proper operation, the oscillator clock should be a stable signal). When the Reference PLL is locked to the reference clock frequency input, it generates the signals that control the Data PLL's center frequency. The signals generated by the Reference PLL are Vfb, V3, V4 and V5.

The Reference PLL's phase comparator compares the pulse width of the output of the one shot and the high time of the reference clock ($\overline{WIND}$). If the pulse width of the reference input and the output of the one shot are equal, the Reference PLL is in "Lock". If the pulse width of the reference clock is more than the width of the one shot, the phase comparator activates the pump-down signal. If the width of the one shot is greater than the high time of the reference clock, then it activates the pump-up signal (higher the current, shorter the pulse width). The charge pump charges or discharges the loop capacitor, depending upon the state of the pump-up and pump-down signals, to reduce the error.

The data separator design is based on the assumption that the signal input to the reference logic is stable, and not affected by temperature, voltage and process variations. When the Reference PLL is locked, a known relationship exists between the reference signal and the effective "RC" time constant of the one-shot. This "standard" RC is then used in the Data PLL. Figure 2.4 shows the relationship between $\overline{WIND}$, TRIG and the output of the one shot.

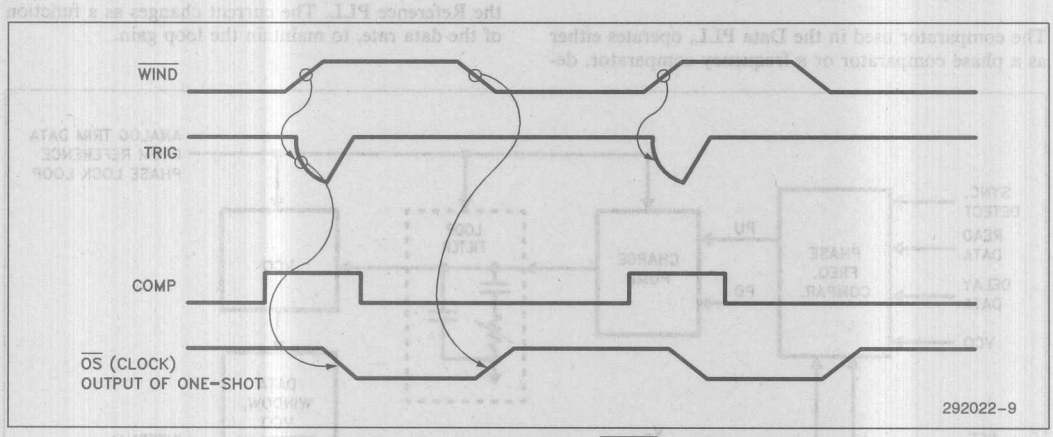


Figure 2.4. Relationship Between $\overline{WIND}$, TRIG, COMP

2.1.3.3 DATA PLL CIRCUIT DESCRIPTION

The Data PLL is the section of the data separator that actually performs the data separation. The data PLL is made up of two major blocks—an analog PLL and sync detect logic. The analog PLL is a classic second order closed loop system.

The elements of the Data PLL are a phase comparator, charge pump, loop filter, transconductance amplifier, current mirror, VCO, frequency divider and some delay logic. The block diagram of the Data PLL is shown in Figure 2.5.

The VCO used in the Data PLL is very similar to the analog one shot used in the Reference PLL. The VCO normally operates at four times the data rate in the MFM mode and eight times the data rate in the FM mode. The center frequency of the VCO is controlled by the Reference loop (V_3, V_4, and V_5) PLL. The output of the VCO is fed into a divider (see the block diagram) which generates the Data Window signal and the Vfbo signal. The divider circuit divides the VCO input by four or two, depending upon whether it is FM or MFM, to generate the Data Window signal. The Vfbo signal is always twice the frequency of the Data Window signal.

The comparator used in the Data PLL, operates either as a phase comparator or a frequency comparator, depending upon the input. If the input to the comparator is the reference frequency, then the frequency comparator mode is selected. The phase comparator mode is selected when reading data. The sync detector logic controls the comparator's mode of operation. The sync detector activates the phase comparator mode when it recognizes the valid sync pulses.

The rising edge of the Read Data input enables the circuit for phase comparison. The phase comparator (reading data) compares the phase difference between the delayed read data and the V_{fbo}. If the rising edge of delayed read data occurs first, the pump-up signal is activated and the VCO frequency is increased. If the Vcofb edge occurs before the Delayed read data, then the pump-down signal is activated and the VCO frequency is decreased. When a low to high transition of both the delayed read data and the Vcofb occurs at the same time, the frequency of the VCO is maintained.

The charge pump controls the current flow (in or out) of the loop filter. When the pump-up (PU) signal is active, the current source charges the loop capacitor. When pump-down (PD) is active, the current source discharges the loop capacitor. When both the PU and PD are inactive, the current source is turned off. The current source and the charge pump is controlled by the Reference PLL. The current changes as a function of the data rate, to maintain the loop gain.

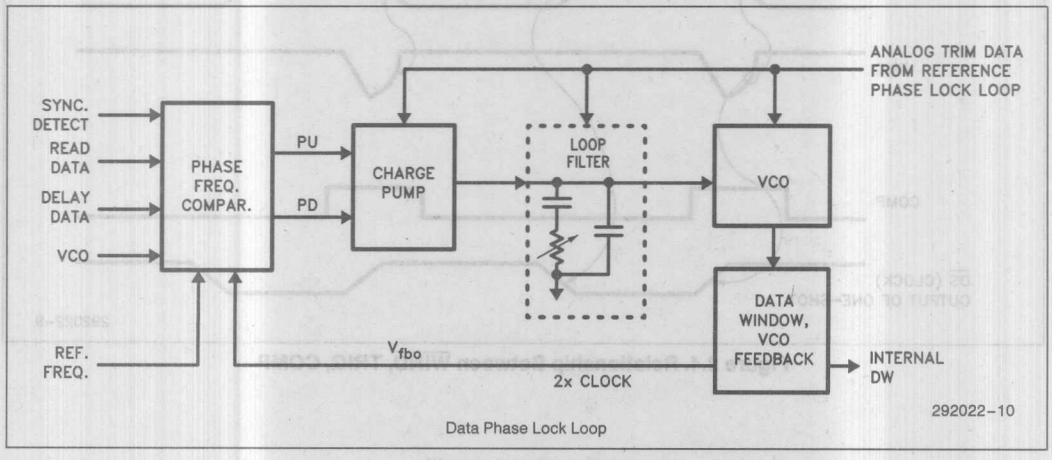


Figure 2.5. Data PLL

During the idle state, the Data PLL is locked to the reference frequency. When the sync detect logic detects eight valid sync bits, the data loop input is switched to the incoming data stream and the VCO is halted and restarted in phase with the read data signal. This reduces the time required to achieve lock.

The VCO requires up to 64 bit times to achieve the final lock. After processing the read data, the Data PLL switches back to the reference input frequency.

2.1.4 Drive Select and Motor Enable Logic

Figure 2.6 shows the circuitry required for generating the drive select and motor enable signals. The 74LS156 is used to decode the DS0, DS1 and the MOTOR signals from the 82072 to generate four drive select and motor enable signals.

The FDC's MOTOR logic provides a programmable motor enable signal. If a drive is not currently selected or the MOTOR signal is not on, the drive select and the MOTOR signals are activated the programmed time before the drive is accessed.

2.1.5 Drive Status Polling

Following hardware reset, the 82072 defaults to the polling enabled mode. When enabled, the disk controller polls the drives periodically while in the idle state. The idle state is after the last byte in the result phase has been read and before the first byte for the command phase has been written. The disk controller selects each drive and checks the ready signal. If the drive has changed its ready signal, it generates an interrupt. The host must respond to this interrupt by executing a Sense Interrupt Status command.

If the polling mode is disabled, the 82072 does not check for changing READY status except on drives currently involved in some disk operation. The polling mode may be disabled or enabled by issuing a CONFIGURE command (see Chapter 4).

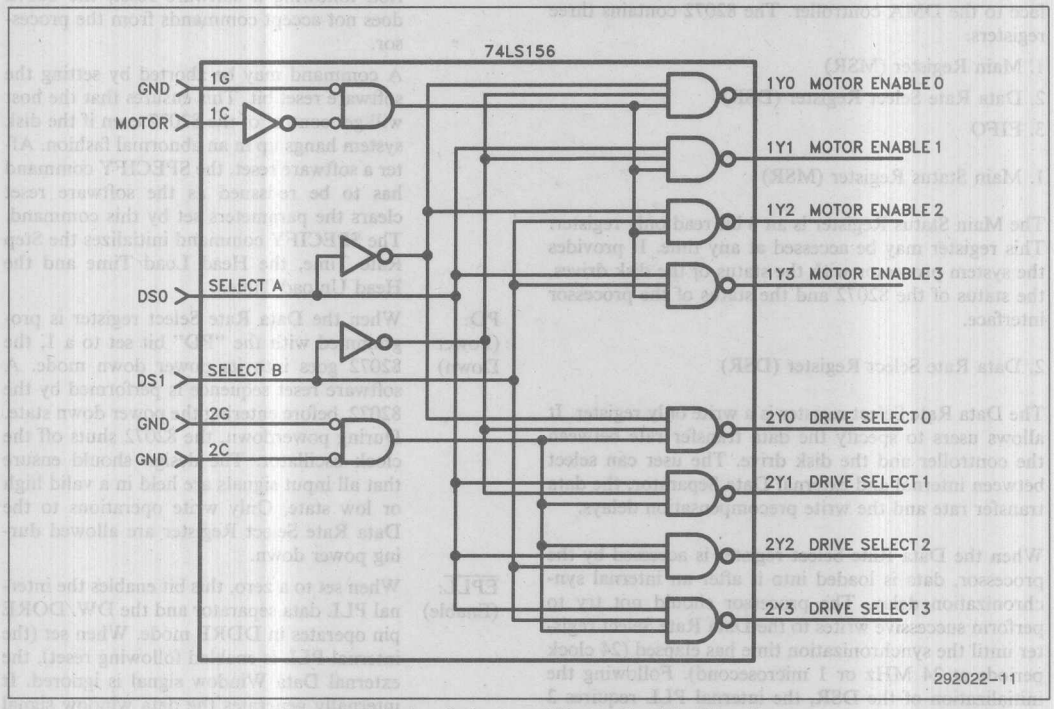


Figure 2.6. Drive Select and Motor Enable Circuitry

CHAPTER 3
FUNCTIONAL DESCRIPTION

The CHMOS III 82072's bus interface has been optimized to interface to the 10 MHz 80186 and the 10 MHz 80286 microprocessors without waitstates. Due to the conventional nature of the hardware signals between the 82072 and the processor, it can easily interface to other microprocessors. This chapter describes the processor interface and the data transfer mechanism between the processor and the disk drive. The emphasis is not on any particular command, but on the data transfer process.

3.1 PROCESSOR INTERFACE

The 82072 system interface consists of a Parallel Interface Unit (PIU) and several registers. The PIU has an 8 bit bi-directional data bus and handles all of the data transfers to/from the system bus. Handshaking signals are provided by the 82072 which makes it easy to interface to the DMA controller. The 82072 contains three registers:

1. Main Register (MSR)
2. Data Rate Select Register (DSR)
3. FIFO

1. Main Status Register (MSR)

The Main Status Register is an 8 bit read only register. This register may be accessed at any time. It provides the system processor with the status of the disk drives, the status of the 82072 and the status of the processor interface.

2. Data Rate Select Register (DSR)

The Data Rate Select register is a write only register. It allows users to specify the data transfer rate between the controller and the disk drive. The user can select between internal and external Data Separator, the data transfer rate and the write precompensation delays.

When the Data Rate Select register is accessed by the processor, data is loaded into it after an internal synchronization delay. The processor should not try to perform successive writes to the Data Rate Select register until the synchronization time has elapsed (24 clock periods at 24 MHz or 1 microsecond). Following the initialization of the DSR, the internal PLL requires 2 milliseconds to stabilize at the new frequency (if you are selecting a new data transfer rate). During this stabilization delay, only head positioning commands should be issued. This register should not be altered during the execution phase of data transfer commands, otherwise data errors would occur. The contents of the DSR register are not affected by software reset. The functions of the various bits in the DSR register are explained below:

SWR	PD	$\overline{\text{EPLL}}$	PRE-COMP			DRATESEL	
7	6	5	4	3	2	1	0

SWR: (Software Reset) When this bit is set to '1', it enables the software reset of the 82072. When enabled, the 82072 internally holds the reset active for 12 to 15 clock cycles (~4 microseconds) depending upon the state of the internal state machine. The processor must wait for the "Request for Master" (RQM bit in the Main status Register) bit to be set before issuing any commands. During the initialization period following a software reset, the 82072 does not accept commands from the processor.

A command may be aborted by setting the software reset bit. This ensures that the host will get control of the 82072 even if the disk system hangs up in an abnormal fashion. After a software reset, the SPECIFY command has to be re-issued as the software reset clears the parameters set by this command. The SPECIFY command initializes the Step Rate Time, the Head Load Time and the Head Unload Time.

PD: (Power Down) When the Data Rate Select register is programmed with the "PD" bit set to a 1, the 82072 goes into its power down mode. A software reset sequence is performed by the 82072, before entering the power down state. During powerdown, the 82072 shuts off the clock oscillator. The design should ensure that all input signals are held in a valid high or low state. Only write operations to the Data Rate Select Register are allowed during power down.

$\overline{\text{EPLL}}$: (Enable) When set to a zero, this bit enables the internal PLL data separator and the DW/DORE pin operates in DDRE mode. When set (the internal PLL is enabled following reset), the external Data Window signal is ignored. It internally generates the data window signal that is used to extract data bits from the serial clock and data bit stream. If this bit

is set to 1, exernal data separation circuitry is required and should provide the Data Window input.

DRATESEL (Data Rate Select): Data Rate Select programs both the read and write data rates. For single density (FM) mode, the data rates are one half the values stated for double density (MFM) mode.

Table 3.1. Supported Data Rates

DRATESEL	DATA RATE	
	MFM	FM
11	1 Mbps	ND*
00	500 Kbps	250 Kbps
01	300 Kbps	150 Kbps
10	250 Kbps	125 Kbps

* Not defined

PRECOMP: The 82072 has write pre-compensation circuitry that internally adjusts the write data pulses before sending it to the disk drive.

The 82072 defaults to PC-AT compatible pre-compensation values when the DSR register is programmed with PRE-COMP bits (bits 2–4) set to zeroes. The pre-compensation value chosen depends upon the programmed data transfer rate (bits 0–1). The default pre-compensation values are illustrated in Table 3.1.1 below.

Table 3.1.1. Default Write Pre-Compensation Delays

Data Rate	Pre-Compensation Delay
1 Mbps	41.67 ns
500 Kbps	125 ns
300 Kbps	125 ns
250 Kbps	125 ns

Upon a hardware reset, the DSR will be configured based on the DDRE setting. When DDRE is tied high, the DSR is configured as follows upon power up and reset:

Table 3.2

SWR	PD	$\overline{\text{EPLL}}$	PRE-COMP			DRATESEL	
7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0

Default Values: 250 Kbit/sec, Internal data separator, Precompensation = 125 ns

When DDRE is tied low, the DSR retains the current data rate and precompensation values:

Table 3.2.1

SWR	PD	$\overline{\text{EPLL}}$	PRE-COMP			DRATESEL	
7	6	5	4	3	2	1	0
0	0	0	X	X	X	X	X

Default values: X = current value, Internal data separator

In the case of the software reset, if EPLL is "0" and DDRE is "1", the Data Rate select and precompensation field will be reset to "00010". It should be noted that under these conditions, a second write to the DSR without a software reset is needed if a different data rate or precompensation value is required.

3. FIFO

The 82072 has a sixteen byte FIFO with a programmable threshold which greatly improves the data transfer mechanism while providing buffering between the serial channel and the system bus. During the command phase, the FIFO is disabled to retain compatibility with the 8272A and to provide proper handling of the "invalid command" condition. A command/parameter byte must be sent after polling the Main Status register to ensure that the controller is ready to accept a byte of data. During the execution phase, the FIFO improves the data transfer rate in two ways. First, the DMA requests to the processor are minimized by fine tuning the FIFO threshold (threshold can be set with the CONFIGURE command. See Chapter 4 for details on the CONFIGURE command) to match the system bus latencies. Second, the host side of the chip does not have to wait for serial side which transfers data at a much slower rate.

The Main Status register, the Data Rate Select register (DSR) and the FIFO can be selected with different combinations of the $\overline{\text{RD}}$, $\overline{\text{WR}}$ and A0 pins as shown below.

Table 3.3. 82072 Register Set

A0	$\overline{\text{RD}}$	$\overline{\text{WR}}$	Function
0	0	0	Illegal
0	0	1	Read Main Status Register
0	1	0	Write to Data Rate Select Register
0	1	1	Data Bus is Tri-stated
1	0	0	Illegal
1	0	1	Read Data from FIFO
1	1	0	Write Data to FIFO
1	1	1	Data Bus is Tri-stated

3.2 PRINCIPLES OF OPERATION

Each command is initiated by a multi-byte transfer from the processor to the 82072 (the transferred bytes

completion of the command specification, the 82072 automatically executes the command. The results after the command execution is also a multi-byte transfer from the controller to the processor. Because of the multi-byte interchange of information between the processor and the controller, it is convenient to consider the command execution as having three phases:

1. Command Phase: The CPU transfers all the information required to perform the command. The 82072 automatically enters the command phase after Reset or following the completion of the result phase of the previous command.

2. Execution Phase: The 82072 executes the command. It enters the execution phase immediately after receiving the last command parameter. The execution phase normally ends after the last byte of data is transferred or if an error occurs.

3. Result Phase: Upon completion of the command, status information is made available to the host processor. After the host reads all the result bytes, the 82072 re-enters the command phase and is ready to accept the next command.

3.2.1. Command Phase

The 82072 is in the command phase after all of the result bytes of the previous command have been read or following RESET. Each command is initiated by a multi-byte transfer from the processor to the 82072. The command and parameter bytes have to be sent to the 82072 in the order shown in the COMMAND SET (82072 Data Sheet). That is, the command code must be sent first and the other parameter bytes must be sent in the prescribed sequence.

During the command phase, bytes are transferred to the 82072 using the Main Status Register to control the direction and timing. Bit 6 (DIO) of the MSR should be cleared and Bit 7 (RQM) should be set before a byte can be written to the FIFO. Many of the commands require multiple bytes, and as a result, the MSR must be checked prior to each byte transfer.

The RQM bit (the MSB in the MSR) goes low following the receipt of the first command byte. The duration that RQM is low, is dependent upon the command being executed. The 82072 holds RQM low and internally sets up the operation to be performed. It sets RQM high when it is ready to accept the next parameter byte. tomatically enters the next phase as defined by the command definition.

3.2.2 Execution Phase

All data transfers occur during the execution phase. The data transfer can be performed either in the DMA or the non-DMA mode, as specified in the SPECIFY command.

1. NON-DMA MODE, Transfers from the FIFO to memory:

When programmed to operate in Non-DMA mode, the 82072 activates the INT pin and sets the RQM bit in the main status register, when the FIFO contains the (16 − threshold) number of bytes or after the last bytes of a full sector transfer has been placed in the FIFO. The MSR should be read to verify that the interrupt is for the data transfer. Bits 5 and 7 of the MSR will be set. The INT pin is used by interrupt driven systems, and the "RQM" bit is used by polled systems. The processor must respond to the interrupt and read the data bytes from the FIFO. The 82072 deactivates the INT pin and RQM bit when the FIFO is empty. This process is repeated until the last byte is transferred out of the FIFO.

2. NON-DMA MODE, Transfers from memory to FIFO:

At the start of the data transfer, the INT pin and the RQM bit in the Main Status register are activated sixteen-byte times before the 82072 enters the execution phase (if the selected data transfer rate is 500 Kbps, then the time from the generation of interrupt to the time the first byte is to be written to the FIFO to prevent underrun errors, is (16 x 16 − 1.5) = 254.5 microseconds. 1.5 microseconds is the time required for serial to parallel conversion). This allows a greater tolerance to interrupt service latency. The CPU must respond to the interrupt by writing data bytes to the FIFO. The serial unit empties the FIFO at the serial data rate. If the interrupt service latency is greater than the (serial data rate X FIFO threshold), an underrun will occur. The Interrupt pin stays active until the FIFO becomes full. The INT pin and the RQM bit of the Main Status register are set again when the FIFO has (Threshold) number of bytes remaining in the FIFO. This process is repeated until the last byte is transferred to the FIFO, as indicated by the activation terminal count. The interrupt pin is deactivated upon receiving terminal count. The 82072 enters the result phase when the serial unit has emptied the last byte from the FIFO.

3. DMA MODE, Transfer from FIFO to memory:

When programmed in DMA mode, the 82072 activates the DRQ pin (DMA Request) when the number of bytes in FIFO equals (16 − threshold) number of bytes, or the last bytes of a sector has been placed in the FIFO. The system DMA has to respond to the request by reading data from the FIFO. The 82072 deactivates the DRQ pin, when the FIFO is emptied by the DMA controller.

4. DMA MODE, Transfers from memory to FIFO:

The 82072 activates the DRQ pin upon entering the execution phase of the data transfer command. The DMA controller must respond to this DMA request by activating its $\overline{DACK}$ and $\overline{WR}$ signals, and placing data in the FIFO. DRQ remains active till the FIFO becomes full. DRQ is activated again when the FIFO contains (16-FIFO threshold) number of bytes. The 82072 also deactivates the DRQ pin when the TC pin is activated, indicating that no more data transfers are required by the command. The 82072 enters the result phase after the last byte has been transferred from the FIFO.

In either mode of operation (DMA or the Non-DMA mode), the execution phase ends when a terminal count signal is sensed, or when the last track has been read or written. In addition, if the disk drive is in a not-ready state at the beginning of the execution phase, the not ready flag is set in the status register 3 and the command is terminated. If a fault signal is received at the end of the execution phase, the FDC sets the equipment check flag in the status register 0, and terminates the command.

3.2.3 Result Phase

Following the completion of the disk operation, status information is made available to the processor. The result phase is indicated by the generation of an interrupt. All of the result bytes have to be read to properly terminate a command.

Bit 6 (DIO) and 7 (RQM) must be set before a byte can be read from the FIFO. The RQM and DIO bits will remain set until the last result byte has been read by the processor. After the last byte is read in the result phase, there is a 12 μs delay (@500 Kbps) before RQM becomes ready to accept a new command.

3.3 SETTING THE FIFO THRESHOLD

The transaction involved in bus acquisition and release imply overhead resulting in losing system clocks due to signal propagation delays and arbitration time requirements. For many short DMA bursts, up to 5 clocks are lost due to this switching. On the other hand, long burst transfers imply that other masters will experience long delays in bus acquisition. An optimal FIFO threshold must be selected that improves system performance and, at the same time, ensures that OVERRUN and UNDERRUN errors are avoided.

A. OVERRUN ERROR: An overrun error occurs if the FDC is not serviced before the FIFO overflows. This error occurs only during read data transfers. The timing requirements for preventing overrun errors can be understood by considering an example.

Let us assume that the FIFO threshold has been set to 10. During the execution phase of a read data command, the DRQ line (when programmed in the DMA mode) is active when the FIFO contains (16 − threshold) = 6 bytes. The DMA controller must respond by reading data from the FIFO before the serial unit adds 10 additional bytes to the FIFO. If the DMA response latency is greater than the time taken by the serial unit to assemble the 10 additional bytes, then an overflow problem occurs.

To prevent overrun error, the DMA latency time must be less than

Threshold × the serial data transfer rate.

At 500 Kbps (High Capacity Quad Density Drives) data transfer rate:

$= 10 \times 16\ \mu s = 160\ \mu s.$
$= 1.28 \times 10^3$ bus cycles at 8 MHz.

B. UNDERRUN ERRORS: An underrun error occurs if the FDC is not serviced before the FIFO is completely emptied by the serial unit. This error is encountered only during write data transfers. At the start of write data transfers, during the execution phase, the FDC 82072 activates the DRQ (DMA request) line sixteen byte times before it actually requires a byte of data. This gives ample time for the DMA controller to respond to the DMA request and place data bytes into the FIFO.

The underrun error timing requirements can be understood by extrapolating the overflow example given above.

Let us again assume that the FIFO threshold has been set to 10. During the execution phase of write data transfers, the DRQ line is activated when the FIFO contains less than the Threshold number of bytes. The DMA controller must respond to the DMA request and write data bytes to the FIFO before the serial unit empties the FIFO.

To prevent underrun errors, the DMA response latency must be less than

Threshold × serial data transfer rate.

At 500 Kbps (High Capacity Quad Density Drives) data transfer rate:

$= 10 \times 16\ \mu s = 160\ \mu s.$
$= 1.28 \times 10^3$ bus cycles at 8 MHz.

The 82072's CONFIGURE command provide a means for choosing the optimal setting required by the application.

If the FIFO is set too low, there are longer periods of time between DMA requests, but the DMA controller must be very responsive to DMA requests. This is the desired operating mode when interfacing to a "fast" system, and typically implies the FDC must be given a high DMA priority.

A high threshold value is used with a "slow" system. This allows for a long latency period after a DMA request, but results in more DMA service requests. The optimal setting should be a compromise between the number of times the bus is accessed, and ensuring that there is no overflow/underflow problem. Typically, a low threshold value is chosen when the 82072 is given a lower DMA priority.

The parameters that determine the optimal threshold value are the parallel bus frequency (Fp), serial data transfer rate (Fs), and the bus latency time (in the range between NAmax and NAmin). The ratio of Fp/Fs determines the relationship between how fast the FIFO is filled and how fast it can be emptied. The bus latency time determines how long the 82072 has to wait before being serviced. If Fs is the serial data rate, then Fs/8 is the serial data rate in bytes per second. The data rate of the system depends on the system clock frequency and the number of clock cycles required to transfer one byte of data (n).

In order for the parallel side to keep up with the serial side, the following must hold:

$$(Fp/n)/(Fs/8) > 1$$

This equation assumes that the parallel interface section will read/write data at a faster rate when compared to the serial section.

To arrive at the optimal threshold value, the designer must take into account that it takes a number of clock cycles to acquire the system bus (defining the range to be between NAmin and NAmax, where NAmin is the minimum time required to acquire the system bus and NAmax is the maximum time). To prevent underrun or overrun, the maximum bus acquisition time must be less than the time to fill the FIFO from the threshold limit (for write operation, the maximum bus acquisition should be less than the time to empty the FIFO from the threshold limit).

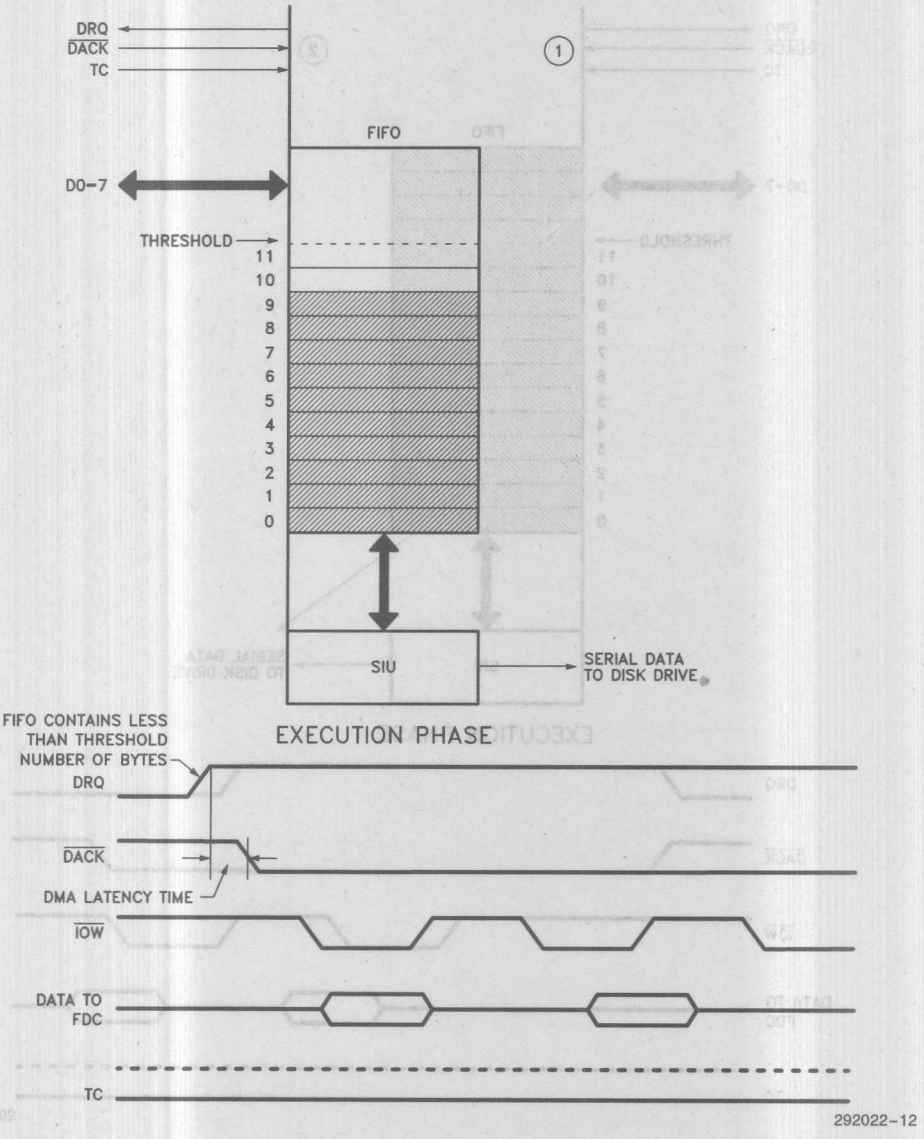
(NAmax/Fp) < (The number of bytes from FIFO threshold to Full (for read))/(Fs/8)

(NAmax/Fp) < (LIMIT)/(Fs)/8;

where LIMIT = FIFO threshold for read/write data transfers.

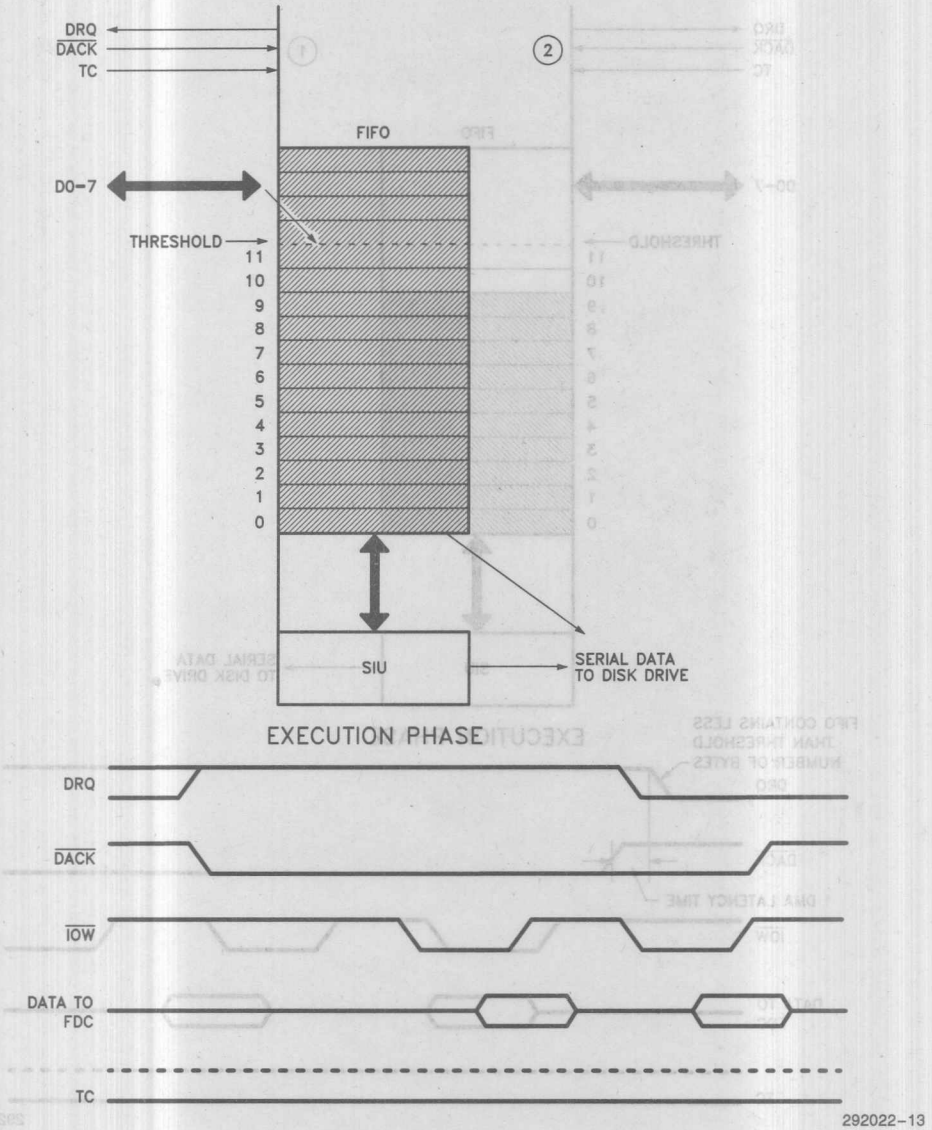
8(NAMax/Fp)/Fs < LIMIT

LIMIT > Fs(NAmax)/8Fp.



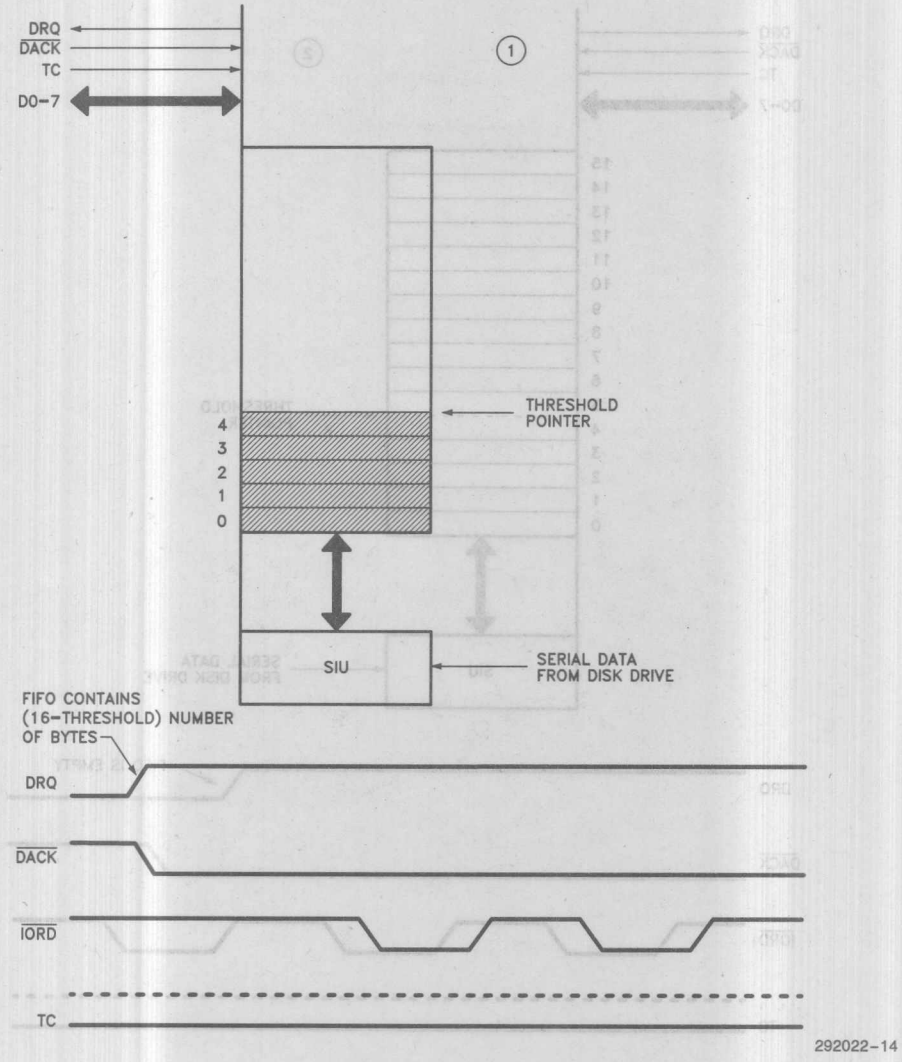
NOTE:
1. During the execution phase, the 82072 generates a DMA request when the FIFO contains less than Threshold number of bytes.

Figure 3.1. FIFO Operation During a Write Command



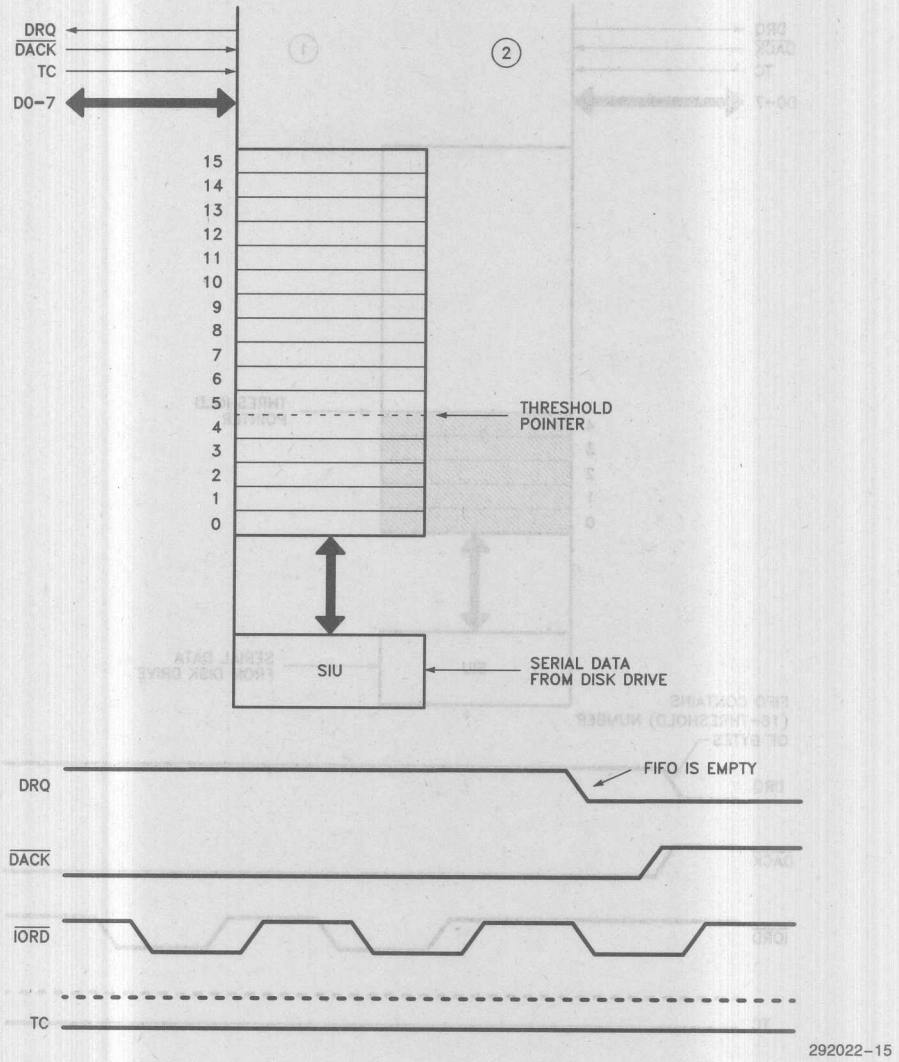
NOTE:
2. DRQ is driven inactive when the FIFO is full or when a terminal count signal is detected

Figure 3.1. FIFO Operation During a Write Command (Continued)



NOTE:
1. During execution phase of read command, the 82072 asserts it's DMA request when the FIFO contains (16 – Threshold) number of bytes.

Figure 3.2. FIFO Operation During a Read Command



NOTE:
2. The 82072 deactivates the DMA request signal when the FIFO is empty

Figure 3.2. FIFO Operation During a Read Command (Continued)

Table 3.4. Minimum Safe FIFO Limit

Fp/Fs	Clocks Between DMA Bursts N1	NAmax	NAmin	FIFO Threshold	
				Min	Max
8	500	100	20	2	8
16	500	100	20	1	12
26.67	500	100	20	1	13
32	500	100	20	1	14

In order to assure that the CPU can have N1 clock cycles between burst cycles, the following conditions must hold: the time it takes the serial unit to fill the FIFO till the programmed threshold plus the minimum bus acquisition time must be greater than the time interval between burst cycles.

$$(16-\text{LIMIT})/(Fs/8) + \text{NAmin}/(Fp) > N1/Fp$$

$$\text{LIMIT} < 16 - Fs(N1-\text{NAmin})/(8Fp)$$

Note that this analysis assumes steady state conditions, and should only be used as a rough guideline in arriving at the optimal FIFO threshold.

3.3.1 Bus Utilization

Bus utilization is defined as the fraction of time that the bus is not busy servicing the 82072 during data transfers. The following analysis also assumes steady state conditions, and should be regarded as a guideline only.

The CPU utilization time is the compliment of the time taken by the CPU to empty the FIFO (tF) divided by the cycle time from FIFO empty to the time it is empty again.

Definition of terms used in the equation

tF = time it takes the system bus to fill the FIFO
tA = bus acquisition time.
FIFO Threshold = fT
BUS UTILIZATION = $1 - tF/(fT/(fp/8) + tF + tA)$

3.4 CONTROLLER/DRIVE INTERFACE

Figure 3.3 is a block diagram of the floppy drive interface required to interface the controller to four high capacity disk drives. Either single or double density disk drives can be supported. No external circuitry is required to generate the write clock. The only external logic needed are a decoder to generate the drive select signals, high current line drivers and Schmitt triggered input gates.

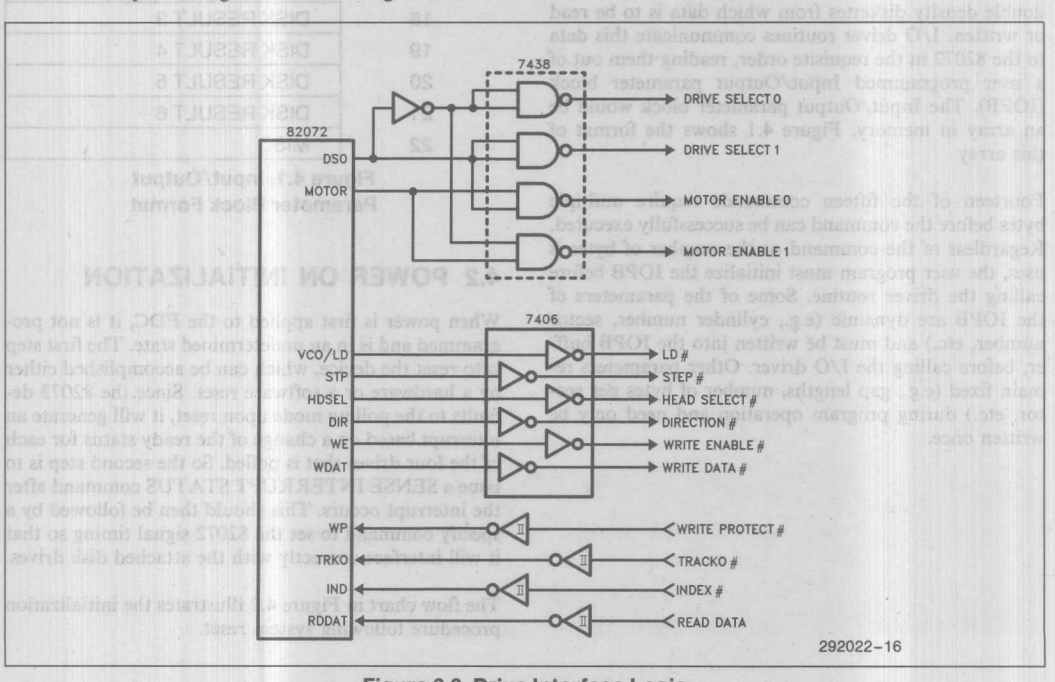


Figure 3.3. Drive Interface Logic

CHAPTER 4
SOFTWARE OVERVIEW

The 82072 is a highly integrated floppy disk controller that supports both single and double density disk storage subsystems. The 82072 supports the IBM 3740 single-density recording format and IBM System 34 double-density recording format. The 82072 accepts and executes high level disk commands such as format a track, read sector and write sector. All data synchronization and error checking is automatically performed by the 82072 to ensure reliable data stroage and subsequent retrieval.

The software required by the 82072 consists of input/output drivers whose functions are to:

1. Initialize the 82072 at power on.
2. Issue commands to the 82072.
3. Handle completion interrupts from the 82072.

4.1 INPUT/OUTPUT PARAMETER BLOCK

Most disk operations require multiple byte transfers to the 82072 before the command can be executed. For example, the program must specify the drive number, the track number, the sector number and the side for double density diskettes from which data is to be read or written. I/O driver routines communicate this data to the 82072 in the requisite order, reading them out of a user programmed Input/Output parameter block (IOPB). The Input/Output parameter block would be an array in memory. Figure 4.1 shows the format of this array.

Fourteen of the fifteen commands require multiple bytes before the command can be successfully executed. Regardless of the command or the number of bytes it uses, the user program must initialize the IOPB before calling the driver routine. Some of the parameters of the IOPB are dynamic (e.g., cylinder number, sector number, etc.) and must be written into the IOPB buffer, before calling the I/O driver. Other parameters remain fixed (e.g., gap lengths, number of bytes per sector, etc.) during program operation and need only be written once.

ADDRESS OFFSET	
0	DMA OPERATION
1	DMA PAGE REGISTER
2	DMA ADDRESS LOW BYTE
3	DMA ADDRESS HIGH BYTE
4	DMA COUNT LOW BYTE
5	DMA COUNT HIGH BYTE
6	DISK COMMAND 0
7	DISK COMMAND 1
8	DISK COMMAND 2
9	DISK COMMAND 3
10	DISK COMMAND 4
11	DISK COMMAND 5
12	DISK COMMAND 6
13	DISK COMMAND 7
14	DISK COMMAND 8
15	DISK RESULT 0
16	DISK RESULT 1
17	DISK RESULT 2
18	DISK RESULT 3
19	DISK RESULT 4
20	DISK RESULT 5
21	DISK RESULT 6
22	MISC

Figure 4.1. Input/Output Parameter Block Format

4.2 POWER ON INITIALIZATION

When power is first applied to the FDC, it is not programmed and is in an undetermined state. The first step is to reset the device, which can be accomplished either by a hardware or a software reset. Since the 82072 defaults to the polling mode upon reset, it will generate an interrupt based on a change of the ready status for each of the four drives that is polled. So the second step is to issue a SENSE INTERRUPT STATUS command after the interrupt occurs. This should then be followed by a specify command to set the 82072 signal timing so that it will interface correctly with the attached disk drives.

The flow chart in Figure 4.2 illustrates the initialization procedure following system reset.

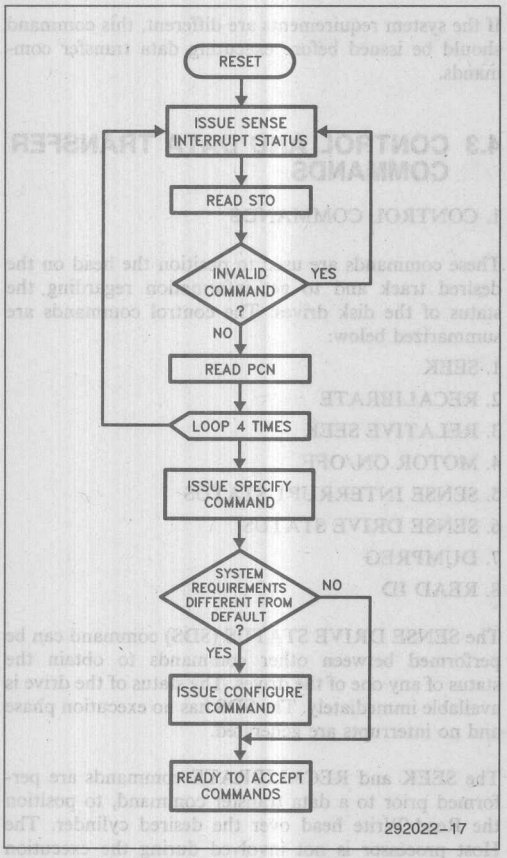


Figure 4.2. Initialization Flowchart

1. SPECIFY COMMAND

This command sets the 82072's signal timings so that the controller will interface correctly to the attached disk drive. The Specify command requires four parameters:

1. Step Rate time (SRT): The 'SRT' defines the time interval between step pulses. Step pulses are used by the disk drive to position the read/write head over the desired cylinder. The step pulse causes the head to move forward or backwards (depending upon the polarity of DIRECTION pin) by one track.
2. Head Load time (HLT): The 'HLT' defines the time interval that the 82072 waits after loading the head before initiating the read or write operation.
3. Head Unload time (HUT): The 'HUT' defines the time interval after the execution phase (of a read or write command) until the head is unloaded.
4. Non-DMA Mode Flag (ND): The 'ND' bit determines if the controller is programmed to operate in the DMA or the non-DMA mode. In the non-DMA mode, the processor is interrupted for each data byte to be transferred. In the DMA mode, the 82072 interfaces to the DMA controller via the DRQ and DACK hand shake signals.

The specify command must be issued prior to performing any disk operations, to define the drive operating characteristics (Refer to the 82072 Data Sheet for timing information). SPECIFY is typically performed following power on initialization. Specify has neither an execution phase nor a result phase.

2. CONFIGURE COMMAND

After system reset, the 82072 defaults to the '8272A' compatible mode. The bits set by the CONFIGURE command default to:

FIFOTHR = 01. Enable FIFO (FIFO threshold is set to "1").

MON = 0000. Motor on delay of 0 seconds.

MOFF = 110. Motor off delay of 5.2 seconds.

PRETRK = 00. The track number beyond which write pre-comp is enabled. Defaults to pre-compensation always enabled.

EIS = 0. No implied seek.

EFIFO = 1. 8272A transfer mode is enabled (i.e., data transfer is performed on a byte by byte basis). The FIFO is enabled with the FIFO threshold set to "1".

POLL = 0. enable drive polling.

HSDA = 0. High Speed Disk Adjust disabled.

The default mode enables the 82072 to be software compatible with the 8272A. After receiving a CONFIGURE command, the 82072 will proceed to initialize its internal registers to enable the desired features.

PRETRACK

The PRETRACK variable allows the programmer to select the track number beyond which the write precompensation is enabled.

FIFOTHR

The 82072 provides a sixteen byte FIFO for a more efficient CPU interface. The FIFO threshold is programmable from 1 to 16 bytes.

MOFF, MON

The motor logic provides a programmable motor signal to the floppy disk drives. If the desired drive is not

currently selected or the MOTOR signal is not on, the drive select and the MOTOR signal are activated the programmed time before the drive is accessed. MON is programmable in increments of rotation times of the disk (0 to 15) for a range of 0 to 3 seconds. Counting rotations makes the delay independent of the data rates.

MOFF is programmed in the same way as MON, giving a range of about 0 to 6 seconds.

If the CPU requests an overlapped SEEK or RECALIBRATE operation, the programmed MON time will be ignored and the step pulses will be issued once the SRT time has elapsed. Refer to the 82072 Data Sheet for additional details.

EIS

Setting EIS to "1" enables implied seek for the READ, WRITE commands. When the implied seek mode is selected, a SEEK command will be executed, before executing the read or write operation. If the seek operation fails, an error flag is set, the 82072 generates an interrupt and enters the result phase of the issued read or write command. The parameter C in the result phase indicates the track number at the time of the abnormal termination. All the remaining status bits are set to zeroes.

EFIFO

This bit controls the mode of data transfers to and from the host system. When this bit is to a "1", the FIFO is disabled, and the 82072 is in the "8272A compatible" mode. In this mode, the data transfers are performed on a byte by byte basis. The 82072 FIFO threshold defaults to one.

POLL

When set to a "1", POLL disables drive polling. The 82072 comes out of reset with polling enabled. When enabled, the 82072 scans all the drives to check if the READY has changed status. Any change of the ready status on any drive will cause an interrupt.

HSDA

High Speed Disk Adjust causes the motor on/off delays to be doubled. This is necessary for disks that rotate at high rates (i.e., 600 RPM vs 300 RPM).

An indepth discussion of the CONFIGURE command can be found in the 82072 Data Sheet.

The CONFIGURE command need not be issued if the system requirements are satisfied by the default mode. If the system requirements are different, this command should be issued before executing data transfer commands.

4.3 CONTROL AND DATA TRANSFER COMMANDS

1. CONTROL COMMANDS

These commands are used to position the head on the desired track and to get information regarding the status of the disk drives. The control commands are summarized below:

1. SEEK
2. RECALIBRATE
3. RELATIVE SEEK
4. MOTOR ON/OFF
5. SENSE INTERRUPT STATUS
6. SENSE DRIVE STATUS
7. DUMPREG
8. READ ID

The SENSE DRIVE STATUS (SDS) command can be performed between other commands to obtain the status of any one of the drives. The status of the drive is available immediately. The SDS has no execution phase and no interrupts are generated.

The SEEK and RECALIBRATE commands are performed prior to a data transfer command, to position the Read/Write head over the desired cylinder. The Host processor is not involved during the execution phase of these commands. At the end of the execution phase of these commands, the 82072 generates an interrupt. A SENSE INTERRUPT STATUS command has to be issued in response to the interrupt.

The MON/MOFF command provides software control of the MOTOR pin. This command has no execution phase and no interrupts are generated.

The DUMPREG command is designed to support system run-time diagnostics and application software development and debug. The DUMPREG command can be issued following the SPECIFY or the CONFIGURE command, to determine if there is proper communication between the Host and the floppy disk controller. This command has no execution phase and no interrupts are generated.

A detailed description of these commands are contained in the 82072 Data Sheet.

2. DATA TRANSFER COMMANDS

The 82072 supports six data transfer commands. They are:

1. Read Data Command
2. Read Delete Data Command
3. Read a Track Command
4. Write Data Command
5. Write Deleted Data Command
6. Format a Track Command

The data transfer commands all require the same parameter bytes and return the same status bytes. The only difference between the data transfer commands are the coding of bits 0–4 (D3–D0) in the first command byte sent to the 82072.

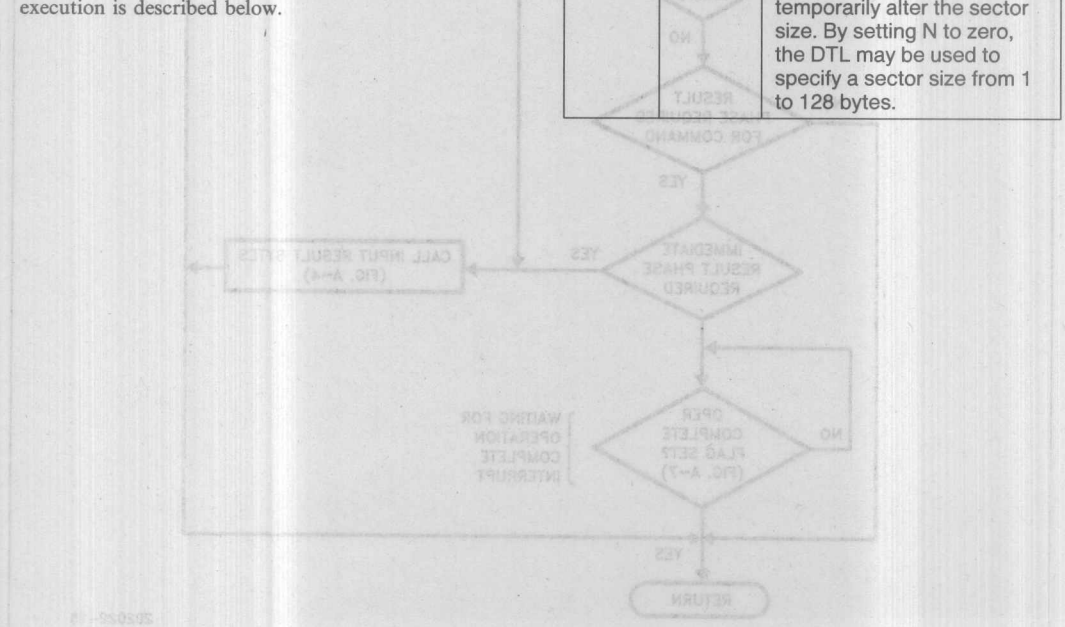
Table 4.1. Data Transfer coding

Command	D3	D2	D1	D0
Read Data	0	1	1	0
Read Deleted Data	1	1	0	0
Write Data	0	1	0	1
Write Deleted Data	1	0	0	1
Read Track	0	0	1	0

The format common to all the data transfer commands and the algorithm for beginning and completing the execution is described below.

The general format of the Command Parameter Block includes the following fields:

Byte #	Symbol	Description
1	HDS & DS0, 1	Head select and drive select.
2	C	Cylinder address. The currently selected cylinder address.
3	H	Selected head address.
4	R	Sector address. Specifies the sector number to read or written.
5	N	Sector size code. The number of data bytes within a sector.
6	EOT	The End of Track. The final sector number of the current track.
7	GPL	Gap Length. Gap 3 size for read or write operation. Gap are regions introduced during disk format. The gaps are used to turn the drives Read/Write head off. This prevents the corruption of the data recorded on the disks.
8	DTL	Special sector size. This parameter is used to temporarily alter the sector size. By setting N to zero, the DTL may be used to specify a sector size from 1 to 128 bytes.



EXECUTING A DATA TRANSFER COMMAND

As mentioned earlier in section 2.4, each command can be considered to have three phases.

1. Command Phase: The processor issues all the command/parameter bytes required to perform the command.
2. Execution Phase: The 82072 performs the operation as instructed.
3. Result Phase: The controller provides the CPU with status information.

The paragraphs to follow describe the three phases with reference to data transfer commands. Figure 4.4 shows the flow chart for the Data Transfer commands.

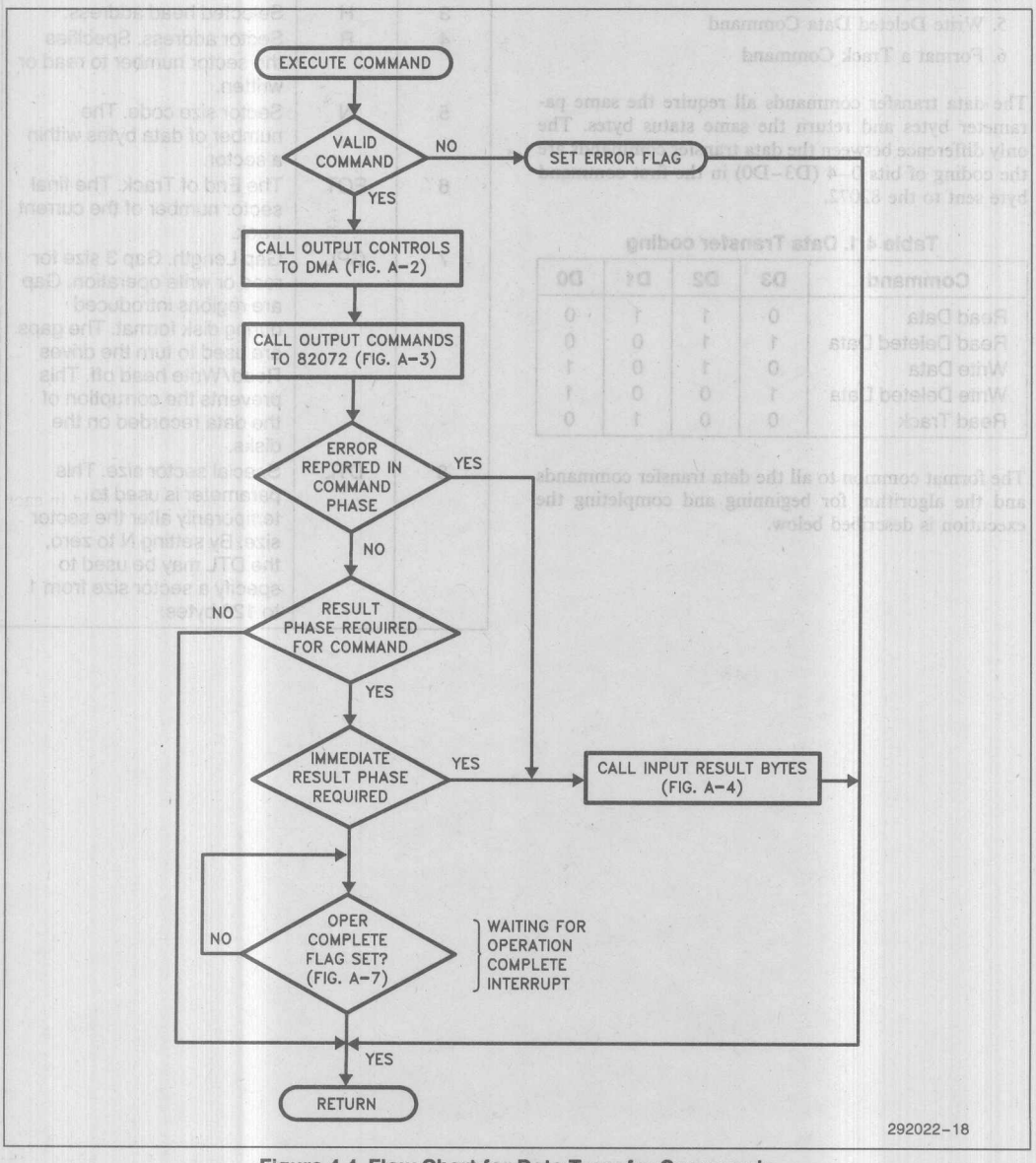


Figure 4.4. Flow Chart for Data Transfer Commands

1. COMMAND PHASE OF THE DATA TRANSFER COMMAND

The data transfer commands consist of a command byte and eight parameter bytes. (Except for FORMAT command, which requires one command byte and five parameter bytes.) The host processor must transfer the command and parameter bytes required to perform the data transfer operation, in the prescribed sequence (refer to the 82072 Data Sheet for the command sequence). The command bytes must be sent after polling the Main Status Register to determine if the FIFO register is available. The handshake mechanism to transfer command bytes is explained in Section 2.4.1. After the last byte of data is sent to the 82072 in the command phase, the 82072 automatically enters the execution phase. In a similar fashion, after the last byte of result is read from the 82072 in the result phase, the result phase is ended and the 82072 reenters the command phase.

The 82072 determines if the first byte sent is a valid command byte. If valid, it continues to request additional bytes. If, on the other hand, the first byte was invalid, the 82072 issues an interrupt and enters the result phase. All the result bytes must be read to successfully terminate the data transfer command.

2. EXECUTION PHASE OF THE DATA TRANSFER COMMANDS

Upon receiving the command byte and 8 parameter bytes, the 82072 loads the head, waits the specified head settling time, and begins reading the ID address marks and the ID fields. When the sector number stored in the ID register matches the sector number read from the diskette, the 82072 starts the actual data transfer. If the current data transfer command being executed is a read from the disk drive, then the controller assembles the serial data bits sent by the disk drive into eight-bit bytes and loads the FIFO. When the FIFO contains the programmed threshold number of bytes, the controller activates its 'DRQ' line (if programmed to operate in the DMA mode) indicating it requires DMA Controller's attention. The DMA controller transfers the data from the FIFO to the system memory. The DRQ line is deactivated when the FIFO is emptied.

The data transfer commands automatically operates in the multi-sector mode, i.e., after completion of read from the current sector, the sector address is incremented by one and the data from the next sector is transferred. Additionally, multi-track operation may be specified (which allows data to be transferred from both sides of the diskette) by setting the 'MT' flag in the command word. The amount of data that can be transferred with a single command depends upon the multitrack flag, the recording density and the number of bytes per sector.

The data transfer is terminated upon receiving a terminal count (TC) through its TC pin or if an attempt is made to read past the 'EOT' (End of Track) specified in the parameter field. Upon receiving the terminal count, the 82072 stops outputting data to the FIFO, but continues to read data from the current sector to check for CRC errors.

ERRORS DURING EXECUTION

The data transfer commands will be terminated if any of the following conditions occur:

1. If the 82072 detects the Index mark twice without finding the requested sector, the 82072 sets the 'sector not found error' flag and terminates the data transfer command. The bits 7 and 6 of the status register ST0 are set to "01".

NOTE:
The controller searches for each sector in a multi-sector operation. Therefore a 'sector not found error' may occur after successful transfer of one or more sectors.

2. If the 82072 reads the deleted data address mark from the disk and the skip flag is not set, the 82072 sets the "control mark" flag (bit 6 in status register 2) and terminates the data transfer command. If the skip flag is set, the 82072 skips the sector with the deleted data address marks, sets the "control mark" flag and reads the next sector. Thus, the 82072 could be forced to skip the sectors with the deleted data address marks, during multi-sector transfers.
3. After reading the ID fields and the data field in each sector, the 82072 checks for CRC bytes. If it finds an incorrect CRC in the ID field, it sets the "data error" flag in the status register 2 and aborts the data transfer. Bits 7 and 6 in the status register ST0 will be set to a '01' respectively.
4. During data transfers between the 82072 and the system, the 82072 must be serviced by the system before an underrun or an overflow condition occurs. These error conditions can be avoided by choosing an optimal FIFO threshold (Chapter 2 shows how to arrive at an optimum FIFO threshold).

3. RESULT PHASE OF THE DATA TRANSFER COMMANDS

After the completion of the data transfer, the 82072 enters the result phase. The 82072 generates an interrupt to the processor. The processor must respond to the interrupt and read the seven result bytes to successfully terminate the data transfer command. The result bytes are read by the "handshake" mechanism described in chapter 2.4.1, and contain status information.

Following the data transfer command, the head is not unloaded until after the head unload time has elapsed. If the host issues another command before the head unload time has elapsed, the head remains loaded. This feature allows subsequent commands to bypass the head load time delay.

CHAPTER 5
APPLICATIONS

This chapter presents three 82072 design examples. The first example illustrates an 82072 in the popular PC environment. It shows a generic floppy disk controller board design that is compatible with the PC/PC-XT. The board can be used as a direct replacement for the existing PC/PC-XT floppy disk controller board. The second design covers the PC-AT. By adding one additional TTL package to the PC/PC-XT design, a 100% PC-AT compatible design can be realized. No software modifications are required.

Additionally, the BIOS can be modified to support the enhanced features of the 82072 such as:

1. Burst data transfers
2. Implied Seek
3. Power Down Mode
4. Relative seek and disk paging
5. 1 Mbps data transfer rate

The third example describes the hardware required to interface the 82072 to the high integration iAPX 186 microprocessor.

Although specific microprocessors are used in each of these examples, the applications can be applied to either MCS85, iAPX86, iAPX88, iAPX 386 or any other processors, provided the necessary hardware changes are made. These applications should serve as a useful guide in illustrating the various procedures in using the 82072.

5.1 EXAMPLE 1. INTERFACING TO THE PC/PC-XT

This example illustrates a generic floppy disk controller board design for the PC/PC-XT.

5.1.1 Interface Requirements

The 82072 based floppy disk controller board occupies a single slot on the I/O channel of the IBM PCs. For clarity, the I/O channel configuration of the PC/PC-XT is used in this example. While the PC-AT's I/O channel configuration differs from the PC/PC-XT, the sub-set of the signals used by the floppy disk controller board is identical.

The floppy disk controller board requires the resources of the processor, DMA and the interrupt controller on the system board. A block diagram of the design is shown in Figure 5.1. As can be seen from the block diagram, the floppy disk sub-system consists of five functional units:

1. Interface between the 82072 card and the I/O Channel
2. Chip-select generation
3. Clock generation
4. DMA interface
5. Drive interface

The implementation of these functional units is discussed in the following paragraphs.

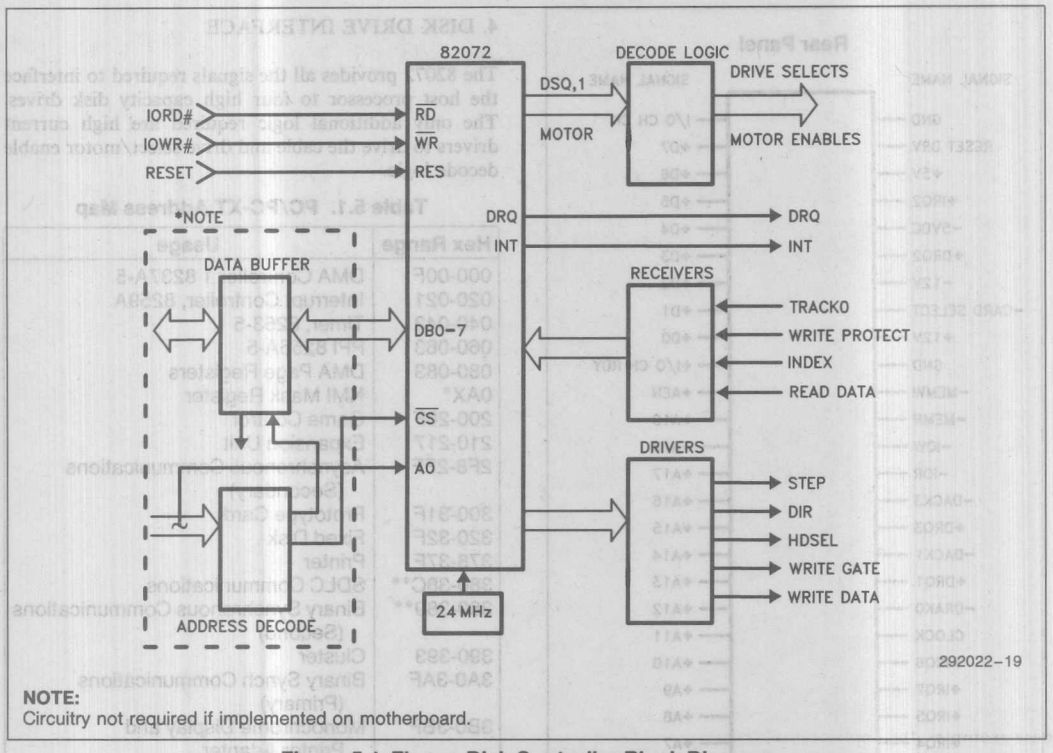


Figure 5.1. Floppy Disk Controller Block Diagram

1. INTERFACING TO THE I/O CHANNEL

The IBM PC/PC-XT has 8 slots on the system board, which allows expansion of the basic system. The expansion slots are electrically identical. The I/O channel contains an 8-bit bi-directional data bus, 20 address lines, 6 levels of interrupt, 3 DMA channels and other control signals required to perform I/O and memory read/write operations. Figure 5.2 shows the signals and the pin assignments for the I/O channel. The existing disk controller boards designed for the PC/PC-XT use interrupt level 6 to get the attention of the processor, and channel 2 of the DMA controller for disk data transfers. To maintain compatibility with the existing floppy disk controller boards, this design example used interrupt level 6 and DMA channel 2.

2. CHIP SELECT AND DATA BUS INTERFACE

Table 5.1. shows the IBM PC's I/O address map. To maintain compatibility with the existing floppy disk controller boards for the IBM PC, the 82072 based floppy disk controller board's I/O port address space was chosen to be 3F0-3F7H. Signals A1 through A9 and AEN (Address Enable) are decoded to generate the chip-select for the 82072.

The PC/PC-XT supports 8 expansion slots. However, because of the fan out constraints, each I/O channel slot signal loading must be limited to only one TTL load. Consequently, to meet this specification, the data bus is buffered through a transceiver. The DT/R (data transmit/receive) line of the transceiver is controlled by a $\overline{\text{IORD}}$ signal. The $\overline{\text{DEN}}$ signal is controlled by "BDSEL" Signal. (Please refer to the PAL equations in Figures 5.4, 5.5). The transceiver is enabled only when the floppy disk controller board is accessed, thereby preventing potential data bus contention.

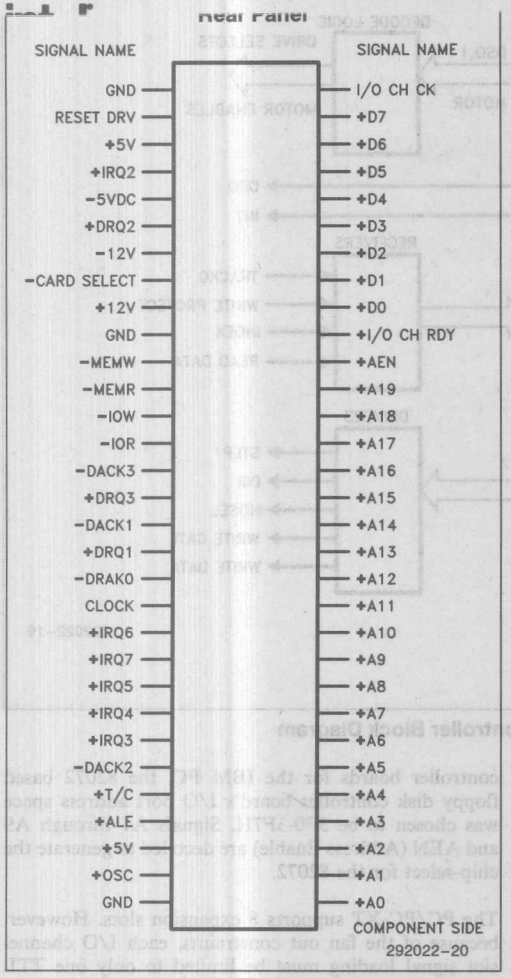


Figure 5.2. I/O Channel Diagram

3. GENERATION

The 82072 requires an external 24 MHz system clock for its operation. The clock input controls the internal operations of the 82072 and its data transfer rate. A 24 MHz fundamental parallel resonant crystal can be attached to the X1 and X2 input pins of the 82072 or the X1 (X2 floats) input can be driven directly by a MOS level clock. This design example uses an external 24 MHz crystal.

The 82072 provides all the signals required to interface the host processor to four high capacity disk drives. The only additional logic required are high current drivers to drive the cable and drive select/motor enable decode logic.

Table 5.1. PC/PC-XT Address Map

Hex Range	Usage
000-00F	DMA Controller 1 8237A-5
020-021	Interrupt Controller, 8259A
040-043	Timer, 8253-5
060-063	PPI 8255A-5
080-083	DMA Page Registers
0AX*	NMI Mask Register
200-20F	Game Control
210-217	Expansion Unit
2F8-2FF	Asynchronous Communications (Secondary)
300-31F	Prototype Card
320-32F	Fixed Disk
378-37F	Printer
380-38C**	SDLC Communications
380-389**	Binary Synchronous Communications (Second)
390-393	Cluster
3A0-3AF	Binary Synch Communications (Primary)
3B0-3BF	Monochrome Display and Printer Adapter
3D0-3DF	Color/Graphics Monitor Adapter
3F0-3F7	Diskette Controller
3F8-3FF	Asynchronous Communications (Primary)

*Upon reset, the NMI into the 8088 is masked off. The mask bit can be set and reset with system programs as follows:
MASK ON: Write I/O address 070H, with data bit 7 equal 0
MASK OFF: Write I/O address 070H, with data bit 7 equal to 1.

**SDLC Communications and secondary Binary Syncnronous communications cannot be used together as their hex addresses overlap.

Motor and Drive Select Logic

The motor enable and drive select signals in this design are derived from an external latch (74LS273). This latch is required to maintain BIOS compatibility with the drive control circuitry. In a normal application where direct motor control by the software is not required, the 82072 can provide the necessary drive select and motor enable signals.

Figure 5.3 shows the circuitry required for generating the drive select and motor enable signals. Drive Select 0 (DS0) and MOTOR signals from the 74LS273 are decoded, using an 74LS04 inverter and 7438 open collector nand gates, to generate two drive select and motor enable signals. The design can easily be upgraded to support four drives by using a decoder/driver of the type 74LS156 with an additional driver.

Head Positioning and Drive Status Circuitry

The step, head select, direction, write enable and write data signal outputs to the disk drive are derived by buffering the STP, HDSEL, DIR, WE and WRDATA from the 82072 through a 7406 open collector driver. The write protect, track 0, index and read data signal inputs are derived by buffering the WP, TRK0, INDX and RDDAT from the disk drive through a 74LS14 Schmitt-trigger inverter. The 82072's internal data separator generates the data window signal which is used by the read hardware to extract data from the serial Read Data stream.

A single 34-conductor cable connects the floppy disk controller board to the disk drives. The last disk drive should be resistively terminated by means of 150Ω resistors pulled-up to +5V.

5. DMA INTERFACE

The 82072 requires a single DMA channel for its operation in the PC/PC-XT environment. The 82072 interfaces to system memory by means of the 8237A-5 DMA controller. Each channel on the 8237A-5 can transfer data throughout the sixteen Megabyte system address space in 64 KB blocks. To support the 20 bit addressing, the IBM PC has a page register for each of the DMA channels. Table 5.2 shows the address generation for the PC DMA channels.

Table 5.2. PC DMA Address Generation

Source	DMA Page Register	8237-5
Address	A19<----------> A16	A15<-----> A0

The processor writes the upper four address bits (A20–A16) to the page register before initiating a data transfer command. When the DMA controller assumes control of the system bus, the contents of the page register are enabled onto the upper four bits of the address bus. The only restriction in the use of this page register is that a single read or write transfer should not cross the 64K memory boundary (FFFFH).

1. OVERVIEW OF THE PC/PC-XT DMA

The IBM PC system board has one 8237A-5 DMA controller. Channel 0 is used for doing the refresh of DRAMs. Channels 1, 2 and 3 are available for add-on boards. The DMA channel assignments for the PC/PC-XT are as follows:

Table 5.3. DMA Channel Assignments

Controller
Chn0-DRAM REFRESH
Chn1-Spare
Chn2-Spare
Chn3-Spare

Since channel 0 of the 8237A-5 DMA controller is used to do the refresh of the DRAMs, all of the remaining channels should be operated in a single-byte transfer mode. In this mode, the channel with the highest priority gets the DMA access. After the DMA cycle is granted, the next channel with the next highest priority gains DMA access. This priority scheme ensures that no single channel can "hog" the bus. More importantly, it ensures that the DRAMs are refreshed once every 15 microseconds, as the refresh channel has the highest priority.

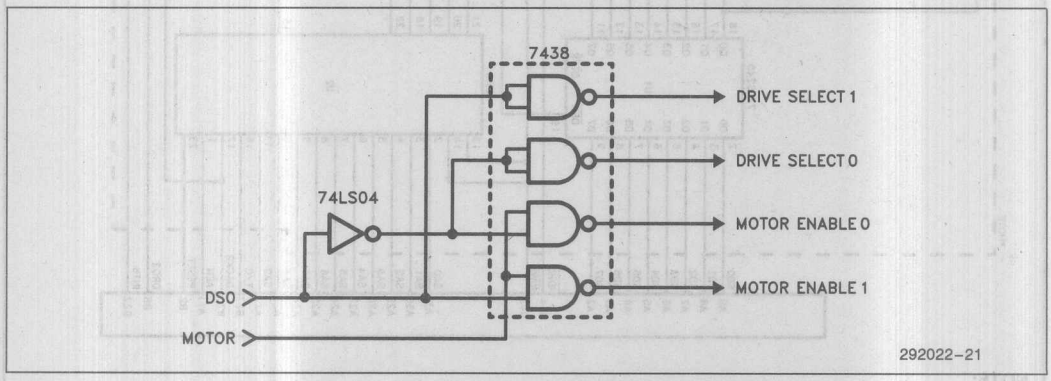
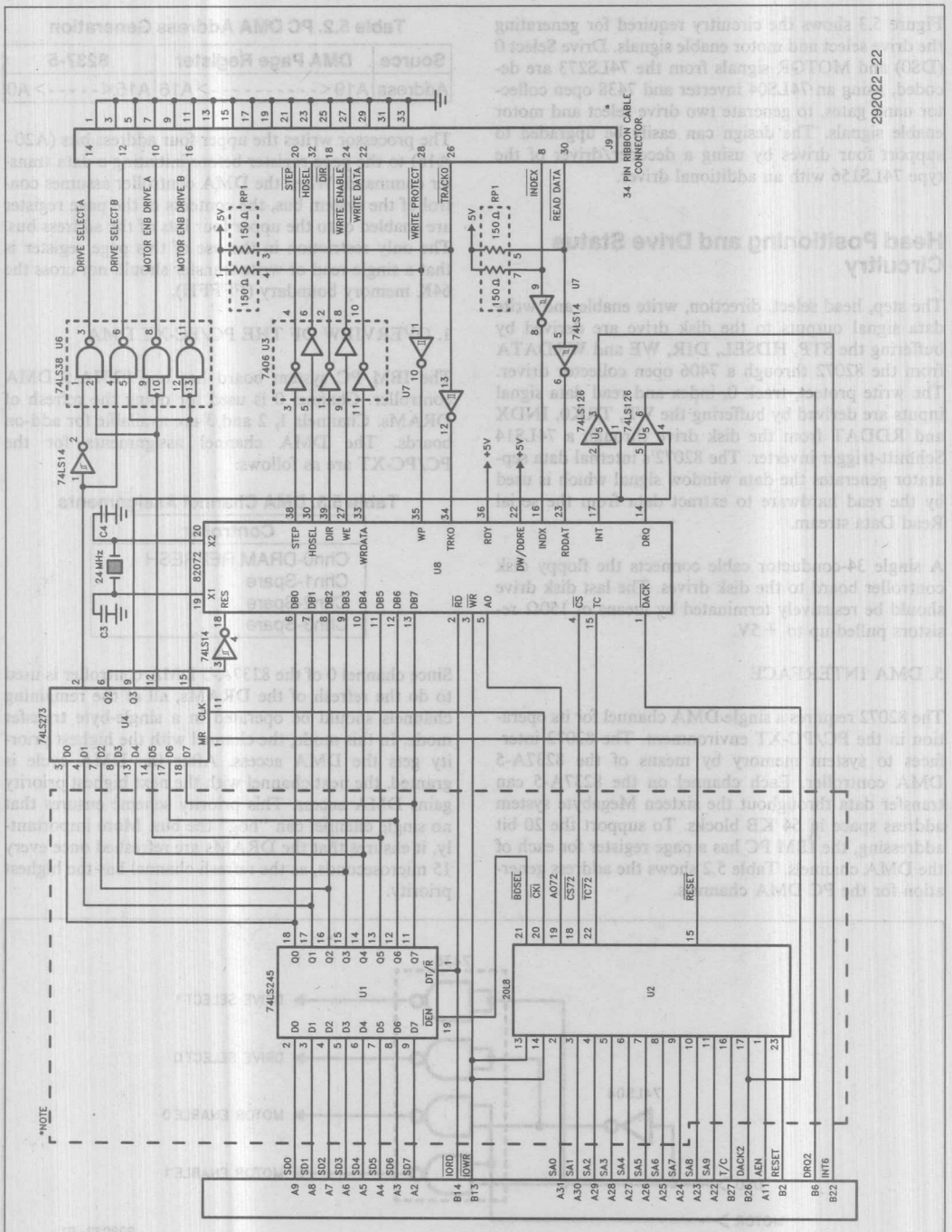


Figure 5.3. Drive Select and Motor Enable Circuit



NOTE:
*Circuitry not required if implemented on motherboard.

Figure 5.4. SBX 072 Floppy Disk Controller Board for PC/PC-XT

```
MODULE SBX072_LOGIC;
TITLE 'SBX 82072 PC/PC-XT COMPATIBLE BOARD DECODE LOGIC' ;
SBX072 DEVICE 'P20L8';
GND, VCC                    PIN 12, 24;
SA9, SA8, SA7, SA6, SA5     PIN 11, 10, 9, 8, 7;
SA4, SA3, SA2, SA1, SA0     PIN 6, 5, 4, 3, 2;
AEN, IOR_, IOW_, RESET      PIN 1, 13, 14, 23;
TC, DACK_                   PIN 16, 17;
BDSEL_, CK1_, A072, CS72_   PIN 21, 20, 19, 18;
TC72, RESET_                PIN 22, 15;

EQUATIONS

" BDSEL_ ENABLES THE 74LS245 TRANSCEIVER FOR WRITES TO 3F2,
" READ/WRITES TO 3F4 AND 3F5, WRITES TO 3F7, OR BY DACK_ FOR DMA
" TRANSFERS

BDSEL_ = !( !DACK_ # ( !AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4 &
         !SA3 & !SA2 & SA1 & !SA0 & !IOW_) # ( !AEN & SA9 & SA8 &
         SA7 & SA6 & SA5 & SA4 & !SA3 & SA2 & !SA1) # ( !AEN &
         SA9 & SA8 & SA7 & SA6 & SA5 & SA4 & !SA3 & SA2 & SA1 &
         SA0 & !IOW_)) ;

" CK1_ ENABLES WRITES TO THE DOR AT 3F2

CK1_   = !( !AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4 & !SA3 &
         !SA2 & SA1 & !SA0 & !IOW_) ;

" A072 IS HIGH FOR READ/WRITES TO THE FIFO AT 3F5 AND LOW FOR
" MSR READS AT 3F4 AND DSR WRITES AT 3F7

A072   = !AEN & SA2 & !SA1 & SA0 ;

" CS72_ SELECT THE 82072 DURING READ/WRITES OF THE FIFO AT 3F5,
" READING THE MSR AT 3F4, OR WRITING TO THE DSR AT 3F4.

CS72_  = !(( !AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4 & !SA3 &
         SA2 & !SA1 ) # ( !AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4
         & !SA3 & SA2 & SA1 & SA0)) ;

" QUALIFY TC BY ACTIVE DACK_

TC72   = (DACK_ & TC) ;

" INVERT RESET FOR LS273 LATCH

RESET_ = !RESET ;

END SBX072_LOGIC
```

Figure 5.5. EPLD Equations for PC/PC-XT Compatible FDC Board

This "byte-by-byte" transfer mode of operation is very slow, as HOLD is dropped after every DMA cycle, and then asserted again for the next cycle. The HOLD/HLDA handshake mechanism or byte-by-byte transfer results in reduced bus bandwidth due to the overhead resulting from latencies in bus acquisition. Burst mode operation is a more desirable approach when using the 82072 but cannot be used here due to the DRAM refresh requirements.

CONCLUSION

The 82072 based floppy disk controller design is 100% compatible with the existing disk controller boards for the PC/PC-XT. Only eight components are required for implementing a disk controller board as opposed to the existing boards that use more than 40 components. Furthermore, the 82072 can interface to the 10 MHz 8088 microprocessors without waitstates (the 8272A introduces two waitstates).

5.2 DESIGNING FOR THE PC-AT

The previous section illustrated the design of a floppy disk controller board for the PC/PC-XT. With the addition of one TTL package, the same design can also be used for the PC-AT.

This example provides a brief overview of the current PC-AT floppy disk controller implementation. It is followed by a discussion of the compatibility issues that arise when interfacing the PC-AT to the 82072 based floppy disk controller board. The section concludes with a discussion of the enhancements provided by the 82072 based disk controller board.

5.2.1 Overview Of The PC-AT

1. Disk Formats

The PC/PC-XT supports four disk formats: single side—8 sectors per track, single side—9 sectors per track, double side—8 sectors per track, and double side—9 sectors per track. The four formats are derived from the number of sides and the number of sectors on each track. The PC-AT has added an additional quad density format, double side—15 sectors per track. Table 5.3.1 shows the data transfer rates associated with the different capacity drives.

Table 5.3.1. The Standard Floppy Disk DOS Formats

Capacity	Density	Drive Speed	Data Rate	Sectors	Tracks
160 K	Single	300 rpm	250 Kbps	8	40
160 K	Single	360 rpm	300 Kbps*	8	40
180 K	Single	300 rpm	250 Kbps	9	40
320 K	Double	300 rpm	250 Kbps	8	40
320 K	Double	360 rpm	300 Kbps*	8	40
360 K	Double	300 rpm	250 Kbps	9	40
1.2 K	Quad	360 rpm	500 Kbps	15	80

* 160K/320K diskettes in 1.2 Mbyte Drives.

2. PC-AT Register Set

To support different capacity drives and the different data transfer rates, the current PC-AT disk controller board incorporates three additional registers: the Data Rate Register (DRR) and the Digital Input Register (DIR) and the Digital Output Register (DOR). These registers are shown in the PC-AT floppy controller Block Diagram, Figure 5.6. The I/O address map for the registers, along with a brief description, are given below:

Table 5.4. I/O Address Map for the PC-AT

Hex Address	Access Type	Description
3F0H	----------	Unused
3F1H	----------	Unused
3F2H	Write	Digital Output Reg
3F3H	----------	Unused
3F4H	Read	Main Status Reg
3F5H	Read/Write	Data Register
3F6H	----------	Unused
3F7H	Write	Data Rate Register
3F7H	Read	Digital Input Reg

PC-AT Data Rate Register (DRR)

The Data Rate Register is used to specify the type of disk drive and the diskette media (if the diskette is double density or quad density diskette). This is a write only register selected on address 3F7H. This value is used by PC-AT floppy disk controller board's data separator circuit to select the data transfer rate and the disk drive spindle speed. The data rate is selected by the BIOS. The decoding for the Data Rate Register is shown below:

Table 5.5. Drive Decoding

D1	D0	Description
1	1	Reserved
1	0	DD Drive DD Diskette
0	1	QD Drive DD Diskette
0	0	QD Drive QD Diskette

NOTES:
DD = Double Density
QD = Quad Density

Additionally, the DRR controls external hardware which generates the "Low Density" (LD) signal. LD is an active high signal that occurs whenever a data transfer rate of 300 Kbps is selected. The PC-AT quad density disk drive uses LD internally to vary the Read/Write head and data channel characteristics. This feature allows the PC-AT quad density disk drive to support double density diskettes.

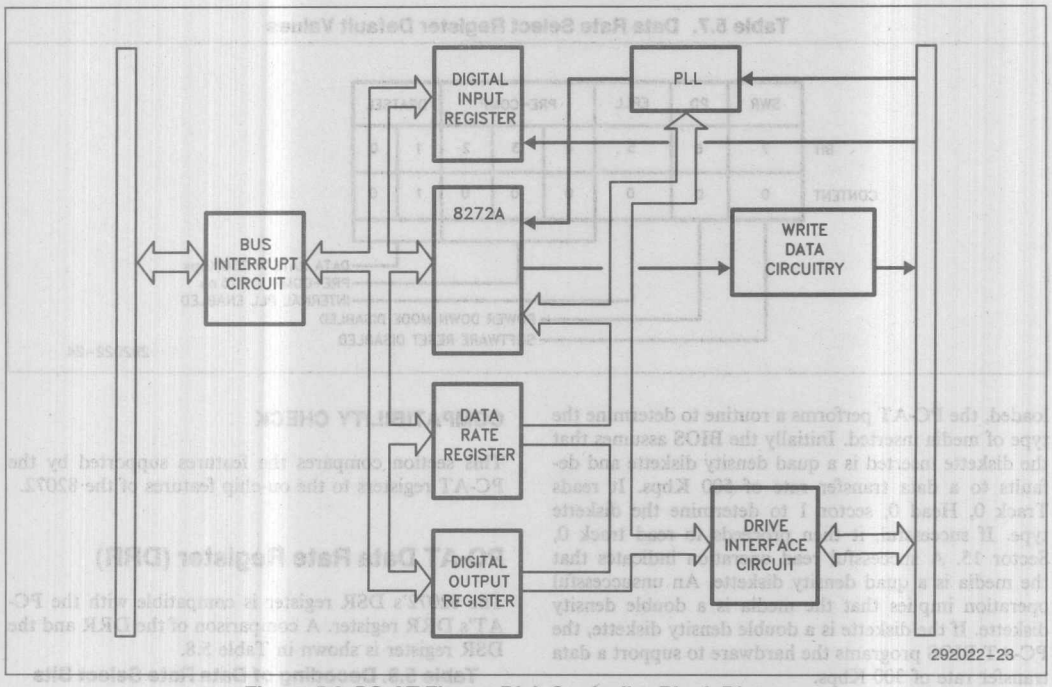


Figure 5.6. PC-AT Floppy Disk Controller Block Diagram

PC-AT Digital Output Register (DOR)

The Digital Output Register is a write only register located at I/O address 3F2H. The individual data bits have the following meanings.

Bits	Definition
0	**Drive Select:** 0 on this bit indicates that drive A is selected, 1 indicates that drive B is selected.
1	**Reserved:** This bit is not used by the hardware. It can be set to a 1 or a 0.
2	**Diskette Function Reset:** When this bit is set to a 1, the diskette reset function is disabled.
3	**Enable Diskette DMA and Interrupts:** When this bit is set to a 1, the DMA and Interrupt lines are enabled.
4	**Drive A Motor Enable:** When this bit is set to a 1, the Motor A enable signal is activated. A timer is dedicated to perform the motor disable function. The timer is initialized by the BIOS, based on the command selected.
5	**Drive B motor Enable;** same as 4.
6	**Reserved;** same as 1.
7	**Reserved;** same as 1.

PC-AT Digital Input Register (DIR)

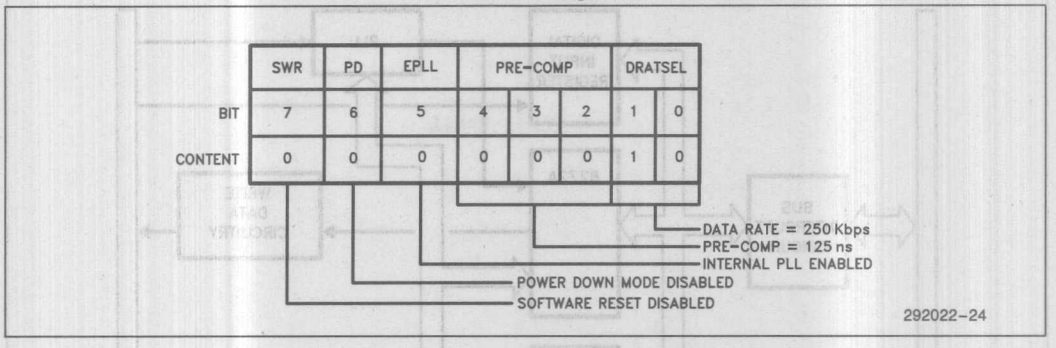
The Digital Input Register is a read-only register located at I/O address 3F7H. Bit 7 is the only bit used by the floppy disk controller. The other bits are used by the hard disk controller for diagnostic purposes. Bit 7 can only be driven when a high capacity drive is active. If set to a "1", it informs the system that the diskette in the disk drive has been changed. Since the PC may have multiple floppy disk drives, the appropriate drive must first be selected, before reading this register. This signal is automatically reset by the drive when a seek operation is performed.

Since bits 0 through 6 of the DIR are reserved by the hard disk controller, it is the user's responsibility to ensure that bit 7 is not driven by the HDC as well. This can easily be done on most hard disk controller boards by changing a jumper that maps out the address area of 3F0H–3F7H to 370H–377H. There is no effect on normal hard disk operations with this change. If this change is not possible with the existing hard disk controller, then DIR support can be implemented by changing the select location of the DIR and modifying the BIOS to support this new location.

3. PC-AT OPERATION

The PC-AT uses a quad density drive that supports both quad density and double density floppy diskettes. Upon power-up or when the quad density disk drive is

Table 5.7. Data Rate Select Register Default Values



loaded, the PC-AT performs a routine to determine the type of media inserted. Initially the BIOS assumes that the diskette inserted is a quad density diskette and defaults to a data transfer rate of 500 Kbps. It reads Track 0, Head 0, sector 1 to determine the diskette type. If successful, it then proceeds to read track 0, Sector 15. A successful read operation indicates that the media is a quad density diskette. An unsuccessful operation implies that the media is a double density diskette. If the diskette is a double density diskette, the PC-AT BIOS programs the hardware to support a data transfer rate of 300 Kbps.

When interfacing to standard double density disk drives, the selected data transfer rate is 250 Kbps.

5.2.2 Compatibility Issues

The 82072 has integrated a software selectable data separator, programmable write pre-compensation logic, Write Clock generation logic and motor on/off logic on-chip. The 82072 has an on-chip clock prescaler that internally divides the 24 MHz clock input to generate the Write Clock. By appropriately setting the data rate selection bits in the DSR, the prescaler value can be changed to obtain the desired Write Clock rate. Additionally, the 82072 has a built-in "Software Reset" feature. This allows the 82072 based disk controller board to support all the features of the PC-AT, without requiring the external data separator or its associated registers. These enhancements were implemented with the objective of maintaining 100% PC-AT compatibility.

Table 5.6. Comparison Between the PC-AT and the 82072

#	Features	82072	PC-AT
1.	Motor Control and Drive Select Ckt	On-chip	External
2.	Programmable Data Transfer Rate	On-chip	External
3.	Software Reset	On-chip	External
4.	DMA and Interrupt enable	External	External
5.	Diskette Change Signal	External	External

COMPATIBILITY CHECK

This section compares the features supported by the PC-AT registers to the on-chip features of the 82072.

PC-AT Data Rate Register (DRR)

The 82072's DSR register is compatible with the PC-AT's DRR register. A comparison of the DRR and the DSR register is shown in Table 5.8.

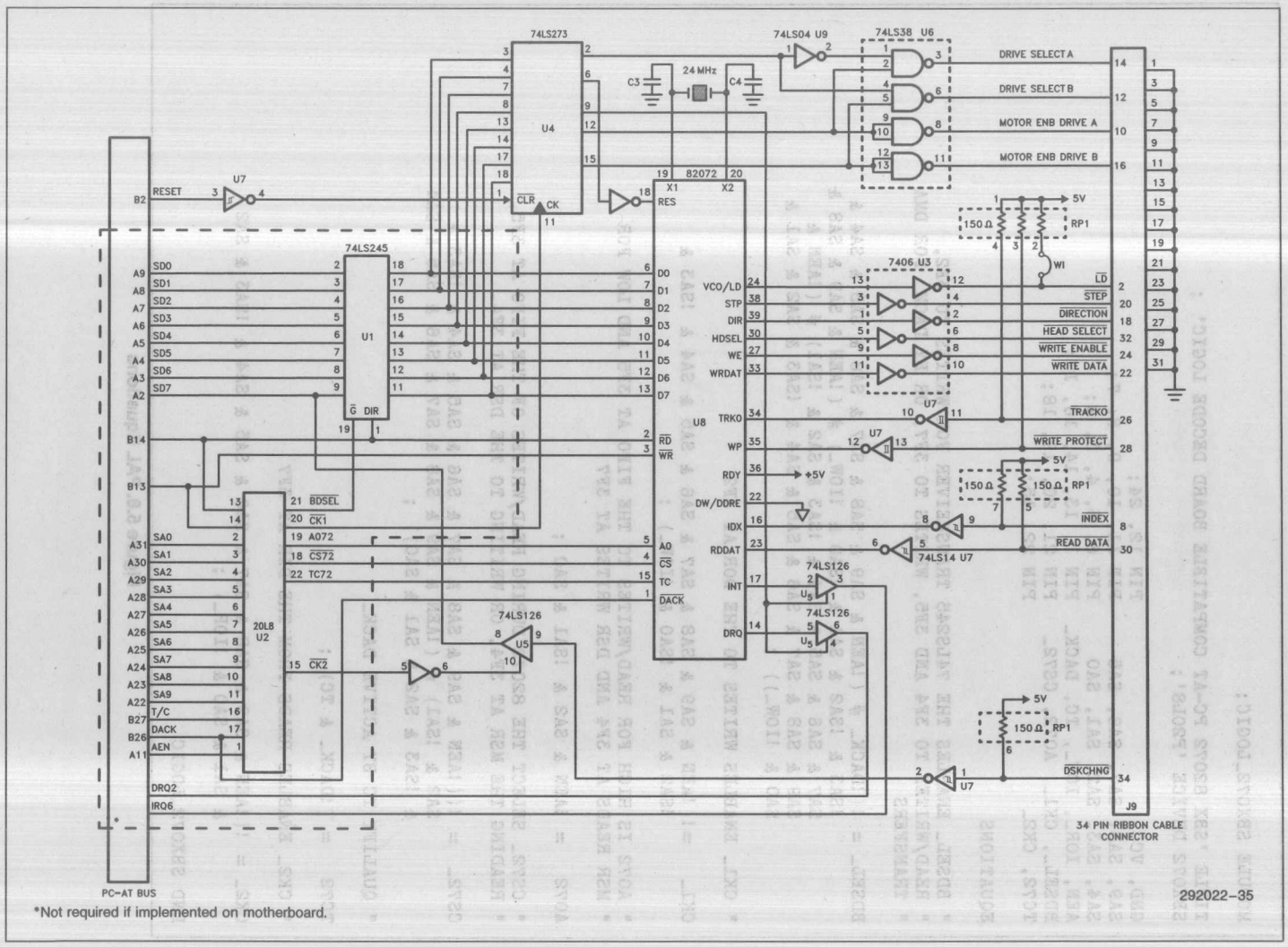
Table 5.8. Decoding of Data Rate Select Bits

Data Rates MFM	PC-AT		82072	
	Bit1	Bit0	Bit3	Bit4
1 Mbps	------	------	1	1
500 Kbps	0	0	0	0
300 Kbps	0	1	0	1
250 Kbps	1	0	1	0

Since the Data rate Select Register (DSR) is internal on the 82072, but treated externally by the PC-AT BIOS, a reset of the 82072 must not reset the DSR. This is possible by connecting the 82072 multiplexed input Data Window/Default Data Rate Enabled (DW/DDRE) to ground. When DW/DDRE is a logic "low", the DSR maintains its current data rate value on a hardware or software reset if the 82072 internal PLL is enabled.

The 82072 supports a multiplexed V_{CO}/LD pin. When the internal PLL is selected by setting the EPLL bit, this pin behaves as the Low Density (LD) output to the disk drive. The 82072 activates the LD signal whenever a data transfer rate of 250 Kbps or 300 Kbps is selected. When the internal PLL is not used, this pin provides the V_{CO} signal to enable an external PLL.

The overall PC-AT compatible design is illustrated in Figure 5.7. The PAL equations map out the registers on board to PC-AT compatible locations. These equations are provided in Figure 5.8.



292022-35

*Not required if implemented on motherboard.

Figure 5.7. SBX 072 Floppy Disk Controller Board for PC-AT

```
MODULE SBX072_LOGIC;

TITLE 'SBX 82072 PC-AT COMPATIBLE BOARD DECODE LOGIC' ;
SBX072 DEVICE 'P20L8';

GND, VCC                    PIN 12, 24;
SA9, SA8, SA7, SA6, SA5     PIN 11, 10, 9, 8, 7;
SA4, SA3, SA2, SA1, SA0     PIN 6, 5, 4, 3, 2;
AEN, IOR_, IOW_, TC, DACK_  PIN 1, 13, 14, 16, 17;
BDSEL_, CK1_, A072, CS72_   PIN 21, 20, 19, 18;
TC72, CK2_                  PIN 22, 15;

EQUATIONS

" BDSEL_ ENABLES THE 74LS245 TRANSCEIVER FOR WRITES TO 3F2,
" READ/WRITES TO 3F4 AND 3F5, WRITES TO 3F7, OR BY DACK_ FOR DMA
" TRANSFERS

BDSEL_ = !(!DACK_ # (!AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4 &
         !SA3 & !SA2 & SA1 & !SA0 & !IOW_) # (!AEN & SA9 & SA8 &
         SA7 & SA6 & SA5 & SA4 & !SA3 & SA2 & !SA1) # (!AEN &
         SA9 & SA8 & SA7 & SA6 & SA5 & SA4 & !SA3 & SA2 & SA1 &
         SA0 & !IOW_)) ;

" CK1_ ENABLES WRITES TO THE DOR AT 3F2

CK1_   =!(!AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4 & !SA3 &
         !SA2 & SA1 & !SA0 & !IOW_) ;

" A072 IS HIGH FOR READ/WRITES TO THE FIFO AT 3F5 AND LOW FOR
" MSR READS AT 3F4 AND DSR WRITES AT 3F7

A072   = !AEN & SA2 & !SA1 & SA0 ;

" CS72_ SELECT THE 82072 DURING READ/WRITES OF THE FIFO AT 3F5,
" READING THE MSR AT 3F4, OR WRITING TO THE DSR AT 3F4.

CS72_  = !((!AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4 & !SA3 &
         SA2 & !SA1) # (!AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4
         & !SA3 & SA2 & SA1 & SA0)) ;

" QUALIFY TC BY ACTIVE DACK_

TC72   = (!DACK_ & TC) ;

" CK2_ ENABLES READS FROM THE DIR AT 3F7

CK2_ = !(!AEN & SA9 & SA8 & SA7 & SA6 & SA5 & SA4 & !SA3 & SA2
       & SA1 & SA0 & !IOR_) ;

END SBX072_LOGIC
```

Figure 5.8. PAL Equations

5.3 NEW FEATURES

If it is the objective of the designer to have no changes to the BIOS, the above design example provides three advantages over the current implementation with minimal effort. It provides the user with:

1. Reduced board space. Only 8 components are required for an 82072 based floppy disk controller board. The current 8272A implementation requires up to 38 components.
2. Reduced power consumption
3. No wait-state interface to the 10 MHz 80286 microprocessor. The 8272A imposes two wait states.

With additional effort, the user can reduce the BIOS and support enhanced features of the 82072 such as:

1. Burst data transfers
2. Implied Seek
3. Power Down mode
4. Relative seek and disk paging
5. 1 Mbps data transfer rate

Burst Data Transfers

During disk transfers between the floppy disk controller and the system, today's floppy disk controllers must be serviced every 13 μs (MFM mode-500 Kbps). This imposes severe timing constraints on the system. During data transfers, the processor is dedicated to the data transfer process. The 82072 overcomes this timing constraint by providing a 16 byte FIFO. The FIFO improves the data transfer mechanism by:

1. The DMA requests to the processor are minimized by fine tuning the FIFO threshold to match system bus latencies.
2. The host side of the 82072 does not have to wait for the serial side which transfers data at a much slower data rate.

Following reset, the FIFO is disabled for compatibility with the 8272A. To improve the system performance, the 82072 CONFIGURE command may be issued to enable the FIFO and select an optimal threshold. The use of the FIFO permits burst data transfers between the 82072 and the system memory. To support burst data transfers, the 8237A mode register has to be programmed to operate in the Demand Transfer Mode (refer to the 8237A data sheet for a description of Demand Transfer Mode).

Implied Seek

An added advantage of using the CONFIGURE command is that the user can also enable the Implied Seek mode. When enabled, the 82072 automatically performs the seek and the sense interrupt status commands before executing the data transfer command, thereby reducing the software overhead required to perform data transfers.

Power Down Mode

The 82072 has a power down mode. If there are known periods of time when the disk controller is not being accessed, the 82072 can be programmed to go into the power down mode. This mode is entered by setting Bit 6 of the DSR register. In this mode, the 82072 consumes less than 1 milliamp of current.

The present cylinder number is maintained while in Power Down mode. However, the status information is cleared. Prior to issuing a Power Down command, it is highly recommended that the user service all pending interrupts.

Relative Seek

This is a new command which enables the 82072 to access more than 256 tracks. The command can be used to partition the diskette into an unlimited number of pages, each page consisting of 256 tracks. This allows the user to exceed the 256 track limit imposed by the IBM format, while maintaining compatibility. Refer to the 82072 Data Sheet or Chapter 1 for further details.

1 Mbps Data Transfer Rate

The 82072 supports data transfer rates up to 1 Mbps. This enables the 82072 to interface to tomorrow's higher capacity disk drives.

5.4 INTERFACING THE 82072 TO THE iAPX 186 DMA CONTROLLER

Although the 82072 interfaces easily to almost any processor, no processor offers as much of the needed functionality as the 80186 or its 8-bit cousin, the 80188. The 80186 is an 8088 object code compatible processor with integrated DMA controller, timers, interrupt controller, chip-select logic, wait state generator, ready logic and clock generation logic on chip.

Figure 5.9 shows a typical 82072/iAPX 186 microprocessor interface. The data lines of the 82072 are connected through buffers to the 80186 AD0–AD7 lines. The 82072, following a read cycle, does not float its output drivers quickly enough to prevent contention with the 186's generated address for the next cycle. To prevent this data bus contention, the data lines of the 82072 are connected through buffers to the 186's AD0–AD7 lines.

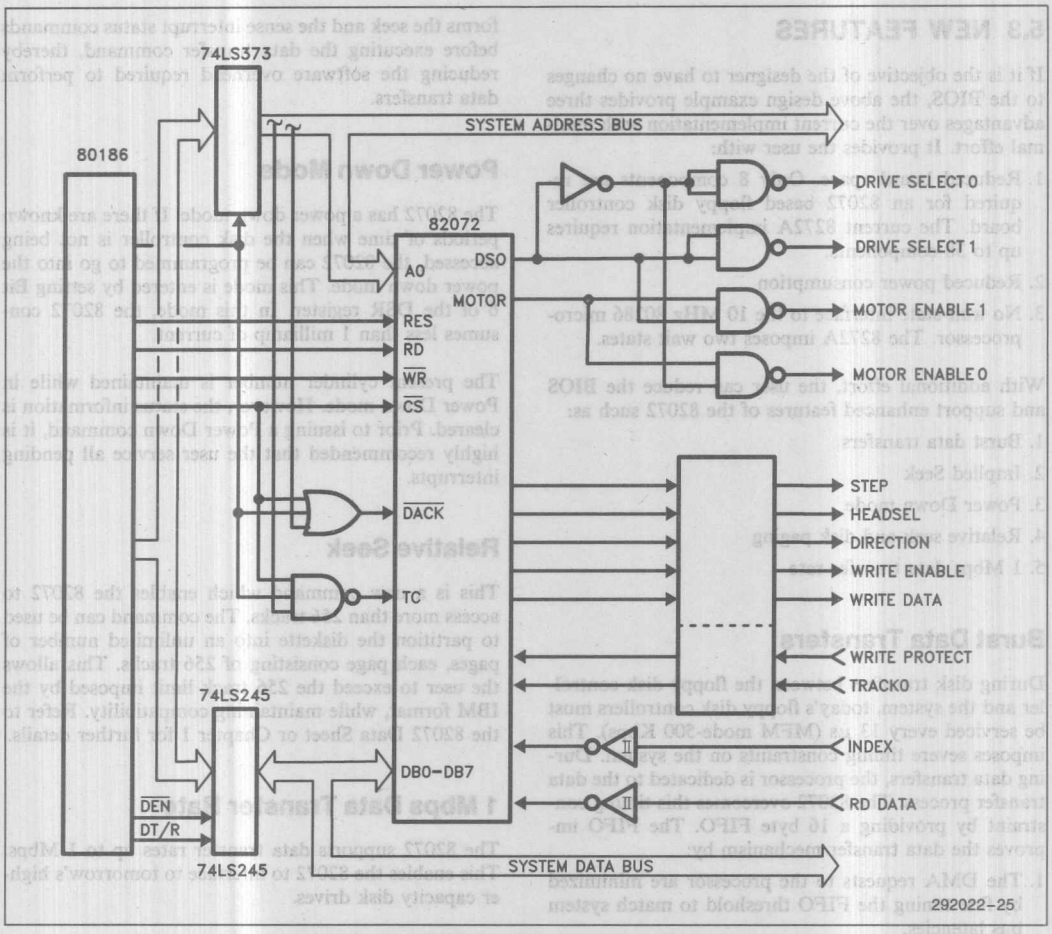


Figure 5.9. INTERFACING to the iAPX 186

The 80186 DMA controller does not provide an explicit DMA Acknowledge or Terminal Count signal required by the 82072. Instead, the 80186 performs a read or write directly to the DMA requesting device. The DMA Acknowledge ($\overline{DACK}$) signal can be generated by decoding an address or merely by using one of the 186 generated chip-select lines. The generation of $\overline{DACK}$ and terminal count (TC) signals are discussed in the following paragraph.

1. $\overline{DACK}$ Generation

The 80186 can generate chip selects for up to seven peripheral devices (PCS0–PCS7). These chip selects are active for seven contiguous blocks of 128 bytes above a programmed base address. The base address is programmable and can only be a multiple of 1 Kbyte.

Let us consider an example to illustrate the generation of $\overline{DACK}$ and TC signals. For this example, assume that PCS4 is connected to the chip-select input of the 82072. The chip-select is activated when an access is made to any I/O location between 3F0–3FFH.

CASE 1: $\overline{DACK}$ generation using a single PCS line

The 80186 activates PCS4 when an access to an I/O location is between 0300H and 03FFH. The $\overline{DACK}$ signal for the 82072 can be generated by an address decode within the region assigned to PCS4. Let us assume for this example that $\overline{DACK}$ is activated when an access is made to the I/O location 0380H.

To generate $\overline{DACK}$ during DMA transfers, the DMA source pointer (for a read transfer) or the destination pointer (for a write transfer) should be initialized to 0380H. The "INC" and "DEC" bits in the 80186 control register must both be set, to prevent the contents of the source or destination pointer from changing.

"ALE" (Address Latch Enable) must be factored into the $\overline{DACK}$ generation circuitry as the addresses are not stable when PCS goes active. This could cause glitches at the output of the $\overline{DACK}$ generation circuitry, as the address lines may change state.

CASE 2: Using an independent chip select line to generate $\overline{DACK}$.

If the system is not using all of the chip select lines provided by the 80186, then the spare chip select line can be used to generate the $\overline{DACK}$ signal for the 82072. This eliminates the extra decode logic required when using a common chip select line. In this type of configuration, PCS4 can be used to generate the chip select for the 82072 and PCS5 can be used to generate the $\overline{DACK}$ signal.

2. Terminal Count Generation

The TC line of the 82072 is driven by a circuit similar to the $\overline{DACK}$ generation circuit. The TC line is used to inform the 82072 to terminate the data transfer. Another method of generating a terminal count signal is by connecting the $\overline{DACK}$ signal to one of the 80186 timers and program the timer to output a pulse after the execution of a certain number of DMA cycles.

APPENDIX A
82072 FLOWCHARTS

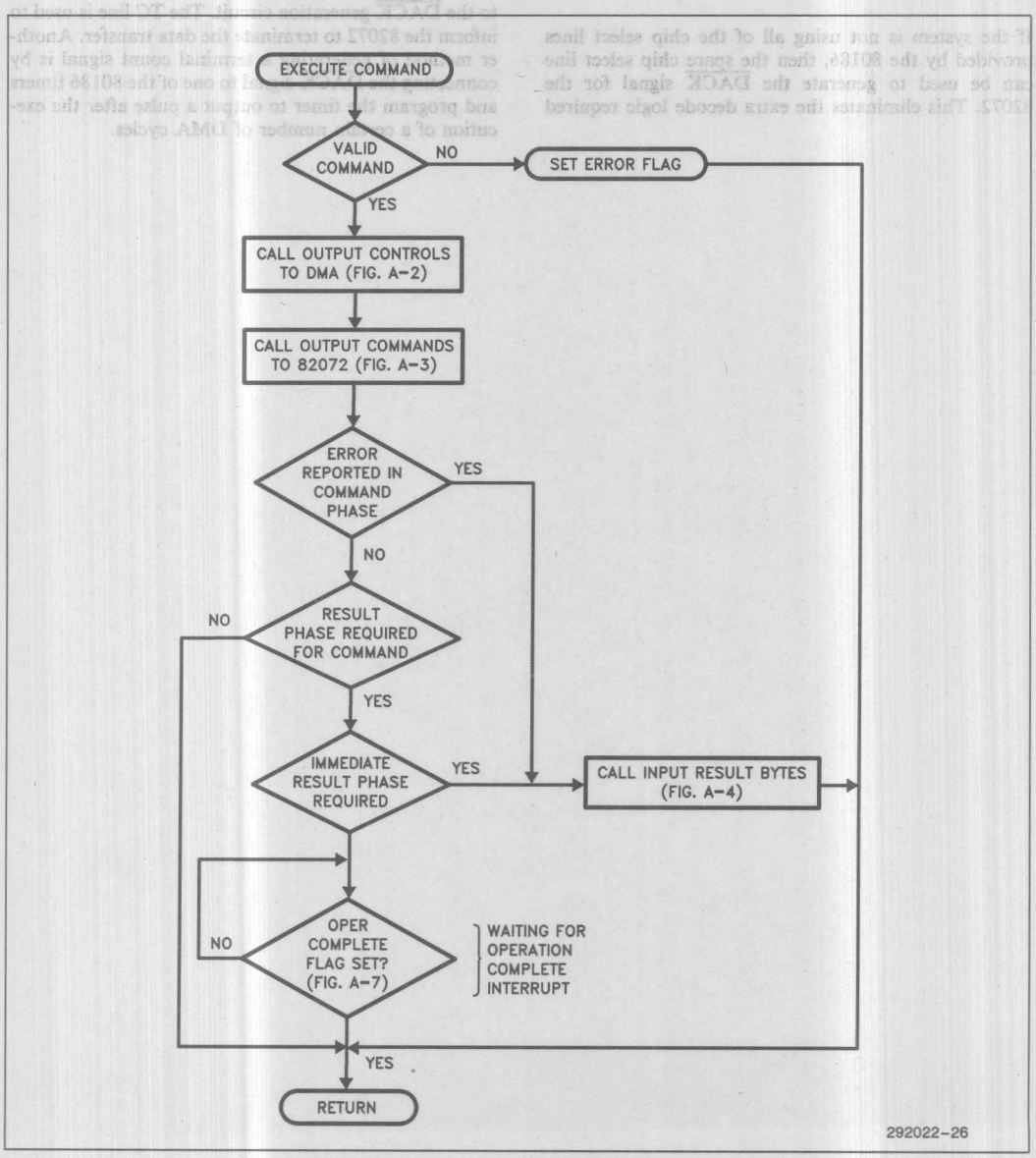


Figure A.1. Generic Command Execution

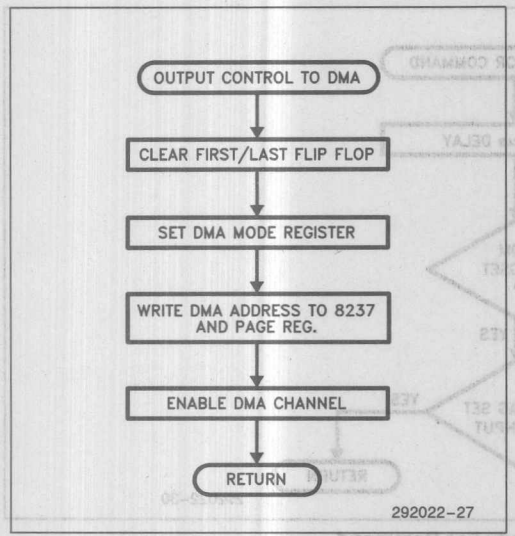


Figure A.2. DMA Initialization

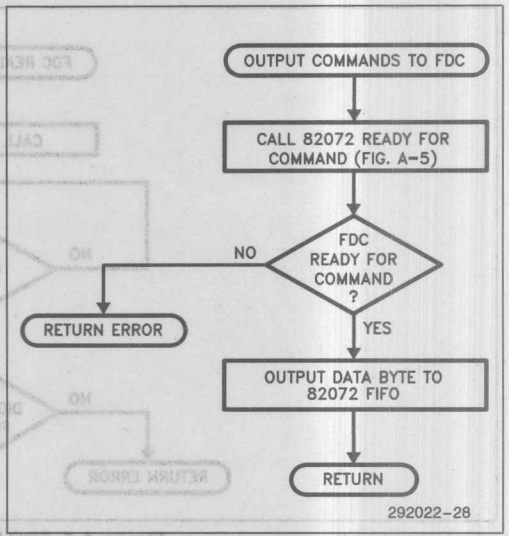


Figure A.3. Output Commands to FDC

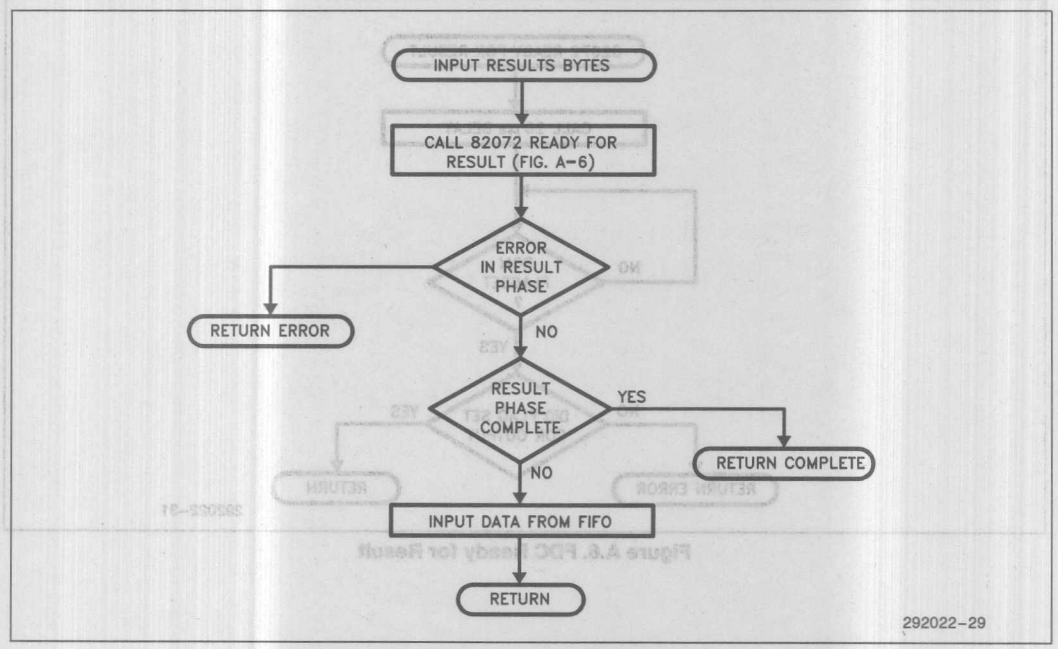


Figure A.4. Input Result Bytes

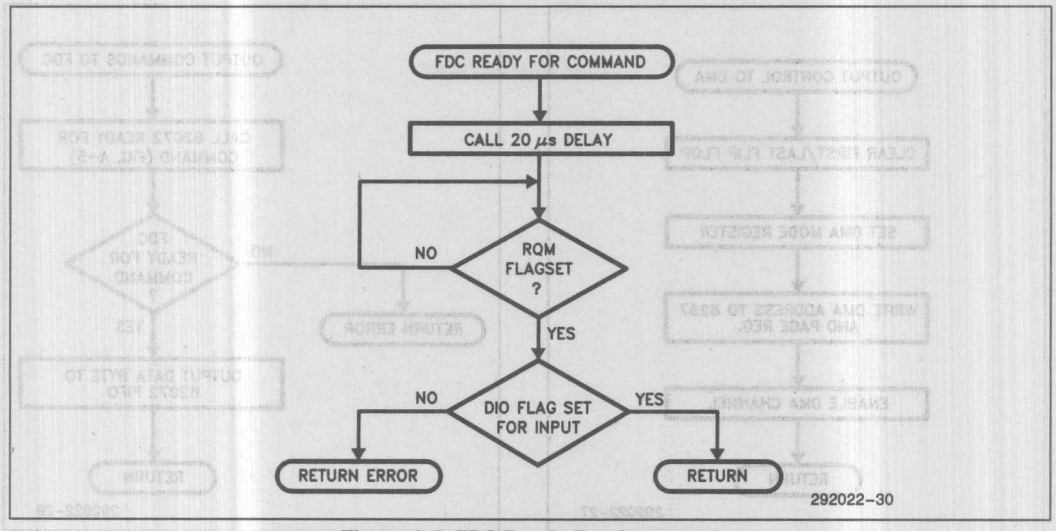


Figure A.5. FDC Ready For Command

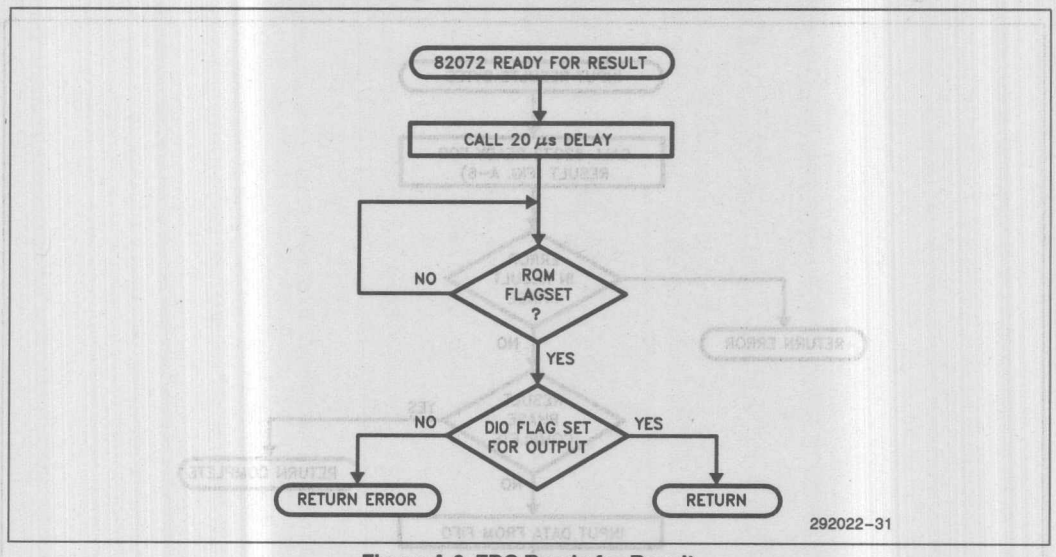


Figure A.6. FDC Ready for Result

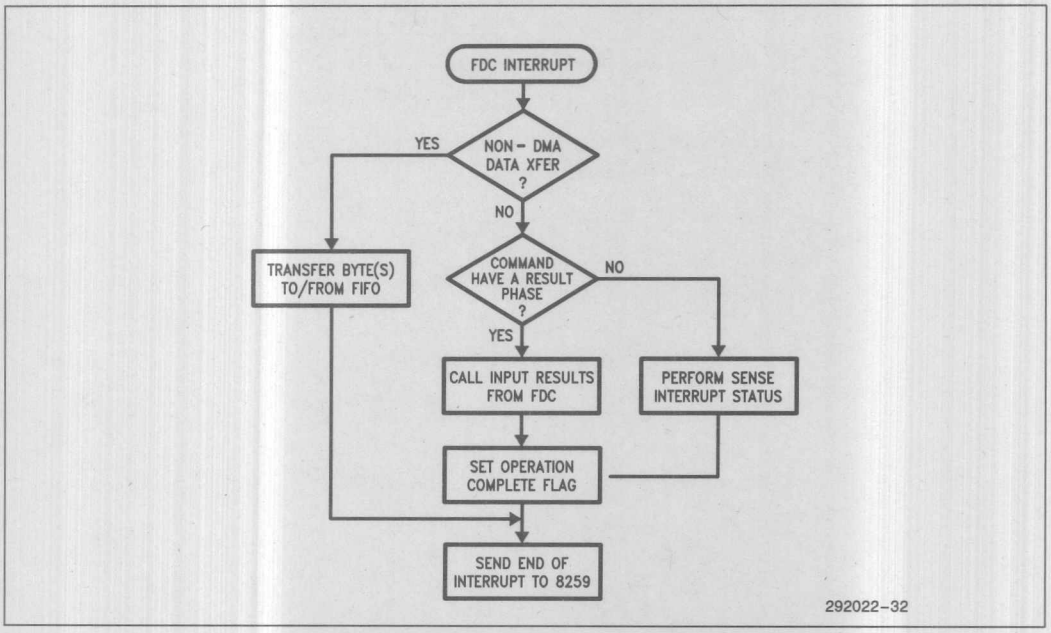


Figure A.7. Interrupt Service Routine

Figure A.7. Interrupt Service Routine

Hard Disk Controllers

4

4 Hard Disk Controllers

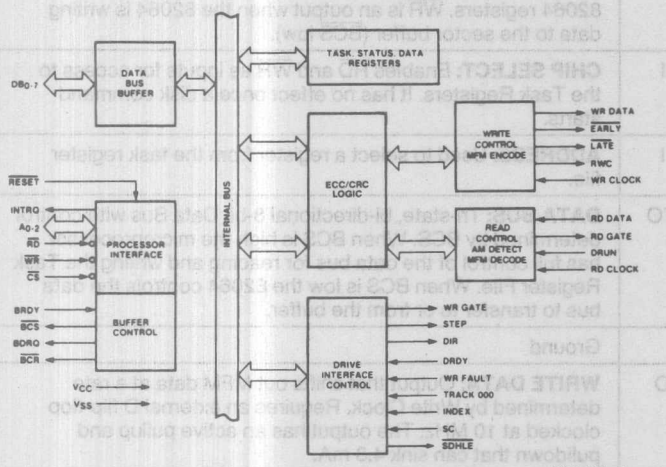
WITH ON-CHIP ERROR DETECTION AND CORRECTION

- Controls ST506/ST412 Interface Winchester Disk Drives
- 5 Mbit/sec Data Transfer Rate
- Compatible with All Intel and Most Other Microprocessors
- High Speed Operation
  - —"Zero Wait State" Operation with 8 MHz 80286 and 10 MHz 80186/188
  - —"One Wait State" Operation with 10 MHz 80286
- Eight High-Level Commands: Restore, Seek, Read Sector, Write Sector, Scan ID, Write Format, Compute Correction, Set Parameter
- Low Power CHMOS III
- On-Chip ECC Unit Automatically Corrects Errors
- 5 or 11-Bit Correction—Span Software Selectable
- Implied Seeks with Read/Write Commands
- Multiple Sector Transfer Capability
- 128, 256, 512 and 1024 Byte Sector Lengths
- Available in 40-Lead Ceramic Dual In-Line, 40-Lead Plastic Dual In-Line, and 44-Lead Plastic Chip Carrier Packages

(See Packaging Spec., Order #231369)

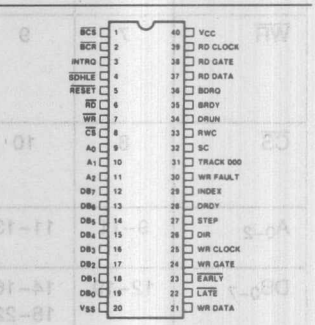
The 82064 Winchester Disk Controller (WDC) with on-chip error detection and correction circuitry interfaces microprocessor systems to 5¼" Winchester disk drives. The 82064 is a CHMOS version of the Western Digital WD2010. It is an upgrade to the Western Digital WD1010A-05 Winchester Disk Controller, and includes on-chip ECC, support for drives with up to 2k tracks, and has an additional control signal which eliminates an external decoder.

The 82064 is fabricated on Intel's advanced CHMOS III technology and is available in 40-lead CERDIP, plastic DIP, and 44-lead plastic leaded chip carrier packages.

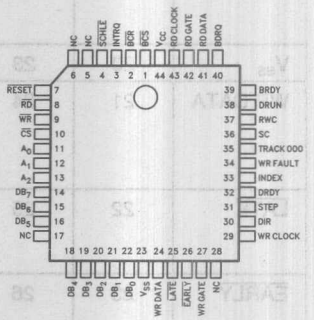


231242-1

Figure 1. 82064 Block Diagram



231242-2



231242-29

Figure 2. 82064 Pinouts

September 1987
Order Number: 231242-005

Table 1. Pin Description

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
$\overline{BCS}$	1	1	O	**BUFFER CHIP SELECT:** Output used to enable reading or writing of the external sector buffer by the 82064. When low, the host should not be able to drive the 82064 data bus, $\overline{RD}$, or $\overline{WR}$ lines.
$\overline{BCR}$	2	2	O	**BUFFER COUNTER RESET:** Output that is asserted by the 82064 prior to read/write operation. This pin is asserted whenever $\overline{BCS}$ changes state. Used to reset the address counter of the buffer memory.
INTRQ	3	3	O	**INTERRUPT REQUEST:** Interrupt generated by the 82064 upon command termination. It is reset when the STATUS register is read, or a new command is written to the COMMAND register. Optionally signifies when a data transfer is required on Read Sector commands.
$\overline{SDHLE}$	4	4	O	$\overline{SDHLE}$ is asserted when the SDH register is written by the host.
$\overline{RESET}$	5	7	I	**RESET:** Initializes the controller and clears all status flags. Does not clear the Task Register File.
$\overline{RD}$	6	8	I/O	**READ:** Tri-state, bi-directional signal. As an input, $\overline{RD}$ controls the transfer of information from the 82064 registers to the host. $\overline{RD}$ is an output when the 82064 is reading data from the sector buffer ($\overline{BCS}$ low).
$\overline{WR}$	7	9	I/O	**WRITE:** Tri-state, bi-directional signal. As an input, $\overline{WR}$ controls the transfer of command or task information into the 82064 registers. $\overline{WR}$ is an output when the 82064 is writing data to the sector buffer ($\overline{BCS}$ low).
$\overline{CS}$	8	10	I	**CHIP SELECT:** Enables $\overline{RD}$ and $\overline{WR}$ as inputs for access to the Task Registers. It has no effect once a disk command starts.
A_{0-2}	9–11	11–13	I	**ADDRESS:** Used to select a register from the task register file.
DB_{0-7}	12–19	14–16 18–22	I/O	**DATA BUS:** Tri-state, bi-directional 8-bit Data Bus with control determined by BCS. When BCS is high the microprocessor has full control of the data bus for reading and writing the Task Register File. When BCS is low the 82064 controls the data bus to transfer to or from the buffer.
V_{SS}	20	23		Ground
WR DATA	21	24	O	**WRITE DATA:** Output that shifts out MFM data at a rate determined by Write Clock. Requires an external D flip-flop clocked at 10 MHz. The output has an active pullup and pulldown that can sink 4.8 mA.
$\overline{LATE}$	22	25	O	**LATE:** Output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders.
$\overline{EARLY}$	23	26	O	**EARLY:** Output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders.

Table 1. Pin Description (Continued)

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
WR GATE	24	27	O	**WRITE GATE:** High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.
WR CLOCK	25	29	I	**WRITE CLOCK:** Clock input used to derive the write data rate. Frequency = 5 MHz for the ST506 interface.
DIR	26	30	O	**DIRECTION:** High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82064's internal comparison of actual cylinder location vs. desired cylinder.
STEP	27	31	O	**STEP:** This signal is used to move the drive head to another cylinder at a programmable frequency. Pulse width = 1.6 μs for a step rate of 3.2 μs/step, and 8.4 μs for all other step rates.
DRDY	28	32	I	**DRIVE READY:** If DRDY from the drive goes low, the command will be terminated.
INDEX	29	33	I	**INDEX:** Signal from the drive indicating the beginning of a track. It is used by the 82064 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.
WR FAULT	30	34	I	**WRITE FAULT:** An error input to the 82064 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.
TRACK 000	31	35	I	**TRACK ZERO:** Signal from the drive which indicates that the head is at the outermost cylinder. Used to verify proper completion of a RESTORE command.
SC	32	36	I	**SEEK COMPLETE:** Signal from the drive indicating to the 82064 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.
RWC	33	37	O	**REDUCED WRITE CURRENT:** Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.
DRUN	34	38	I	**DATA RUN:** This signal informs the 82064 when a field of all ones or all zeroes has been detected in the read data stream by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods.
BRDY	35	39	I	**BUFFER READY:** Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host μP. Only the rising edge indicates the condition.

Table 1. Pin Description (Continued)

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
BDRQ	36	40	O	**BUFFER DATA REQUEST:** Activated during Read or Write commands when a data transfer between the host and the 82064's sector buffer is required. Typically used as a DMA request line.
RD DATA	37	41	I	**READ DATA:** Single ended input that accepts MFM data from the drive.
RD GATE	38	42	O	**READ GATE:** Output that is asserted when a search for an address mark is initiated. It remains asserted until the end of the ID or data field.
RD CLOCK	39	43	I	**READ CLOCK:** Clock input derived from the external data recovery circuits.
V_{CC}	40	44	I	**D.C. POWER:** +5V.
NC	—	5, 6 17, 28		**No Connects**

FUNCTIONAL DESCRIPTION

The Intel 82064 CHMOS Winchester Disk Controller (WDC) interfaces microprocessor systems to Winchester disk drives that use the Seagate Technology ST506/ST412 interface. The device translates parallel data from the microprocessor to a 5 Mbit/sec, MFM-encoded serial bit stream. It provides all of the drive control logic and control signals which simplify the design of external data separation and write precompensation circuitry. The 82064 is designed to interface to the host processor through an external sector buffer.

On-chip error detection algorithms include the CRC/CCITT and a 32-bit computer generated ECC polynomial. If the ECC code is selected, the 82064 provides three possible error handling techniques if an error is detected during a read operation:

1. Automatically correct the data in the sector buffer, providing the host with good information.
2. Provide the host with the error location and pattern, allowing the host to correct the error.
3. Take no action other than setting the error flag.

The Intel 82064 is an enhanced version of the Western Digital WD2010 Winchester Disk Controller. The 82064 has been completely redesigned for Intel's advanced CHMOS III fabrication process, allowing Intel to offer a high quality, low power device while at the same time maintaining complete compatibility with the WD2010.

Enhancements to the basic design include:

Conversion to a CHMOS III fabrication process for low power consumption.

Improvements to the processor interface to provide high-speed "zero wait state" operation with 10 MHz 80186/188 and 8 MHz 80286. High-speed "one wait state" operation with 10 MHz 80286.

The 82064 is completely socket and software compatible with the WD2010 Winchester Disk Controller. As with the WD2010, the 82064 is also socket and software compatible with existing WD1010A-05 designs that do not include external ECC.

INTERNAL ARCHITECTURE

The internal architecture of the 82064 is shown in more detail in Figure 3. It is made up of seven major blocks as described below.

PLA Controller

The PLA interprets commands and provides all control functions. It is synchronized with WR CLOCK.

Magnitude Comparator

An 11-bit magnitude comparator is used to calculate the direction and number of steps needed to move the heads from the present to the desired cylinder position. It compares the cylinder number in the task file to the internal "present position" cylinder number.

A separate high-speed equivelance comparator is used to compare ID field bytes when searching for a sector ID field.

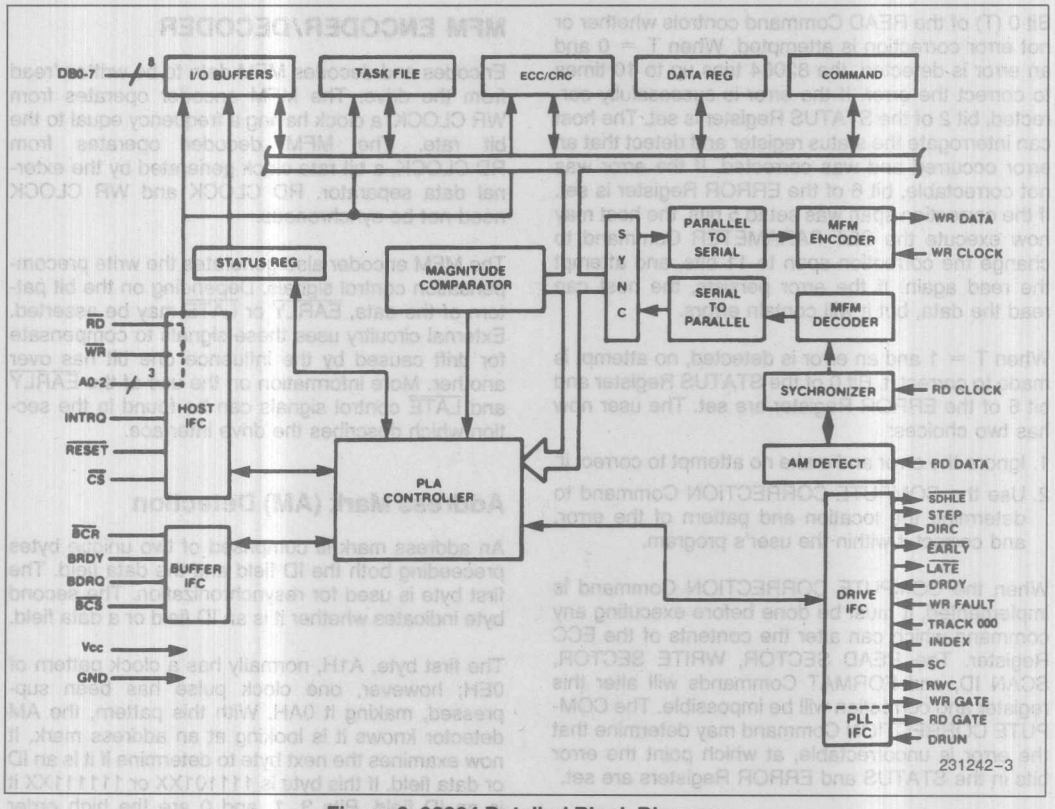


Figure 3. 82064 Detailed Block Diagram

CRC and ECC Generator and Checker

The 82064 provides two options for protecting the integrity of the data field. The data field may have either a CRC (SDH register, bit 7 = 0), or a 32-bit ECC (SDH register, bit 7 = 1) appended to it. The ID field is always protected by a CRC.

The CRC mode provides a means of verifying the accuracy of the data read from the disk, but does not attempt to correct it. The CRC generator computes and checks cyclic redundancy check characters that are written and read from the disk after ID and data fields. The polynomial used is:

$$X^{16}+X^{12}+X^{5}+1$$

The CRC register is preset to all one's before computation starts.

If the CRC character generated while reading the data does not equal the one previously written an error exists. If an ID field CRC error occurs the "ID not found" bit in the error register will be set. If a data field CRC error occurs the "ECC/CRC" bit in the error register will be set.

The ECC mode is only applicable to the data field. It provides the user with the ability to detect and correct errors in the data field automatically. The commands and registers which must be considered when ECC is used are:

1. SDH Register, bit 7 (CRC/ECC)
2. READ SECTOR Command, bit 0 (T)
3. READ SECTOR and WRITE SECTOR Commands, bit 1 (L)
4. COMPUTE CORRECTION Command
5. SET PARAMETER Command
6. STATUS Register, bit 2 - error correction successful
7. STATUS Register, bit 0 - error occurred
8. ERROR Register, bit 6 - uncorrectable error

To enable the ECC mode, bit 7 of the SDH register must be set to one.

Bit 0 (T) of the READ Command controls whether or not error correction is attempted. When T = 0 and an error is detected, the 82064 tries up to 10 times to correct the error. If the error is successfully corrected, bit 2 of the STATUS Register is set. The host can interrogate the status register and detect that an error occurred and was corrected. If the error was not correctable, bit 6 of the ERROR Register is set. If the correction span was set to 5 bits, the host may now execute the SET PARAMETER Command to change the correction span to 11 bits, and attempt the read again. If the error persists, the host can read the data, but it will contain errors.

When T = 1 and an error is detected, no attempt is made to correct it. Bit 0 of the STATUS Register and bit 6 of the ERROR Register are set. The user now has two choices:

1. Ignore the error and make no attempt to correct it.
2. Use the COMPUTE CORRECTION Command to determine the location and pattern of the error, and correct it within the user's program.

When the COMPUTE CORRECTION Command is implemented, it must be done before executing any command which can alter the contents of the ECC Register. The READ SECTOR, WRITE SECTOR, SCAN ID, and FORMAT Commands will alter this register and correction will be impossible. The COMPUTE CORRECTION Command may determine that the error is uncorrectable, at which point the error bits in the STATUS and ERROR Registers are set.

Although ECC generation starts with the first bit of the F8H byte in the data ID field, the actual ECC bytes written will be the same as if the A1H byte was included. The ECC polynomial used is:

$$X^{32}+X^{28}+X^{26}+X^{19}+X^{17}+X^{10}+X^{6}+X^{2}+1$$

For automatic error correction, the external sector buffer must be implemented with a static RAM and counter, not with a FIFO.

The SET PARAMETER Command is used to select a 5-bit or 11-bit correction span.

When the L Bit (bit 1) of the READ SECTOR and WRITE SECTOR commands is set to one, they are referred to as READ LONG and WRITE LONG commands. For these commands, no CRC or ECC characters are generated or checked by the 82064. In effect, the data field is extended by 4 bytes which are passed to/from the sector buffer.

With proper use of the WRITE SECTOR, READ LONG, WRITE LONG, and READ SECTOR Commands, a diagnostic routine may be developed to test the accuracy of the error correction process.

MFM ENCODER/DECODER

Encodes and decodes MFM data to be written/read from the drive. The MFM encoder operates from WR CLOCK, a clock having a frequency equal to the bit rate. The MFM decoder operates from RD CLOCK, a bit rate clock generated by the external data separator. RD CLOCK and WR CLOCK need not be synchronous.

The MFM encoder also generates the write precompensation control signals. Depending on the bit pattern of the data, $\overline{\text{EARLY}}$ or $\overline{\text{LATE}}$ may be asserted. External circuitry uses these signals to compensate for drift caused by the influence one bit has over another. More information on the use of the $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ control signals can be found in the section which describes the drive interface.

Address Mark (AM) Detection

An address mark is comprised of two unique bytes preceeding both the ID field and the data field. The first byte is used for resynchronization. The second byte indicates whether it is an ID field or a data field.

The first byte, A1H, normally has a clock pattern of 0EH; however, one clock pulse has been suppressed, making it 0AH. With this pattern, the AM detector knows it is looking at an address mark. It now examines the next byte to determine if it is an ID or data field. If this byte is 111101XX or 111111XX it is an ID field. Bits 3, 1, and 0 are the high order cylinder number bits. If the second byte is F8H, it is a data field.

Host/Buffer Interface Control

The primary interface between the host processor and the 82064 is an 8-bit bi-directional bus. This bus is used to transmit and receive data for both the 82064 and the sector buffer. The sector buffer consists of a static RAM and counter. Since the 82064 makes the bus active when accessing the sector buffer, a transceiver must be used to isolate the host during this time. Figure 4 illustrates a typical interface with a sector buffer. Whenever the 82064 is not using the sector buffer, the BUFFER CHIP SELECT ($\overline{\text{BCS}}$) is high (disabled). This allows the host access to the 82064's Task Register File and to the sector buffer. A decoder is used to generate $\overline{\text{BCS}}$ when A_{0-2} is '000', an unused address in the 82064. A binary counter is enabled whenever $\overline{\text{RD}}$ or $\overline{\text{WR}}$ go active. The location within the sector buffer which is addressed by the counter will be accessed. The counter will be incremented by the trailing edge of the $\overline{\text{RD}}$ or $\overline{\text{WR}}$. This allows the host to access se-

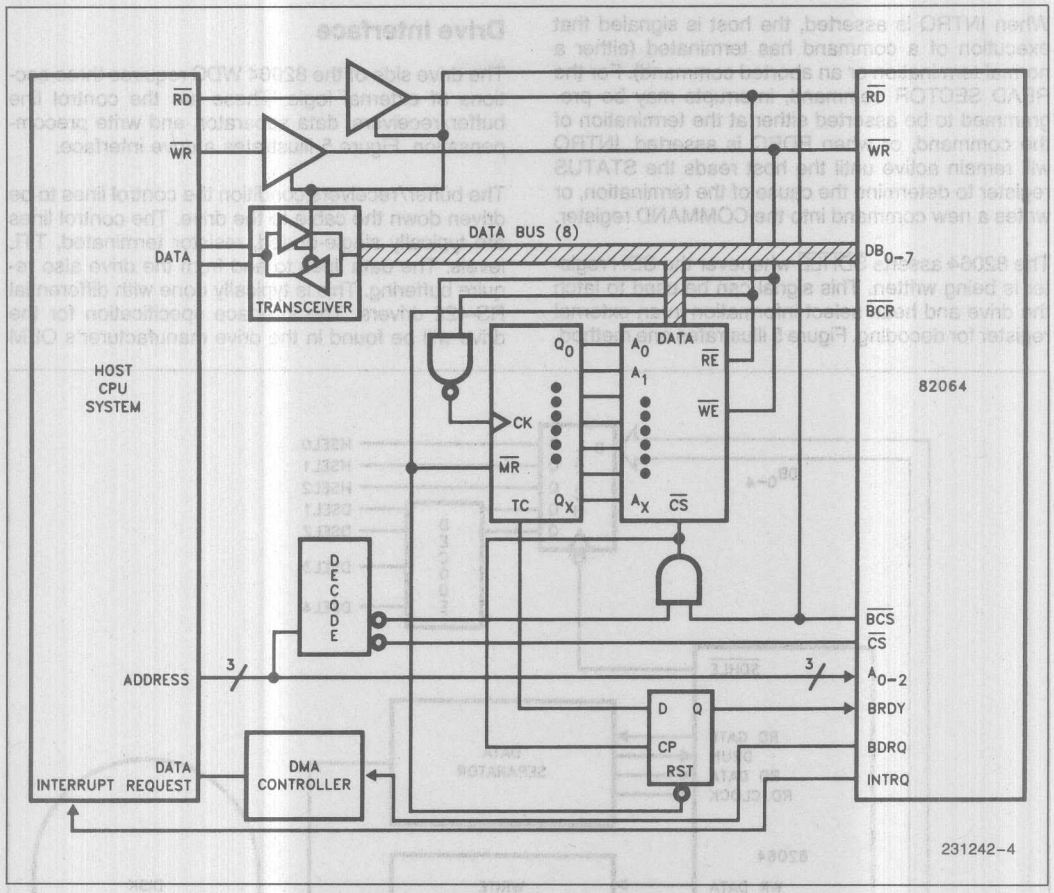


Figure 4. Host Interface Block Diagram

quential bytes within the sector buffer. The decoder also generates a $\overline{CS}$ for the 82064 whenever A_{0-2} does not equal '000', allowing access to the 82064's internal Task Register File while keeping the sector buffer tri-stated.

During a WRITE SECTOR Command, the host processor sets up data in the Task Register File and then issues the command. The 82064 asserts BUFFER COUNTER RESET ($\overline{BCR}$) to reset the counter. It then generates a status to inform the host that it can load the sector buffer with data to be written. When the counter reaches its maximum count, the BUFFER READY (BRDY) signal is asserted by the carry out of the counter, informing the 82064 that the sector buffer is full. (BRDY is a rising edge triggered signal which will be ignored if asserted before the 82064 asserts $\overline{BCR}$.) $\overline{BCS}$ is then asserted, disconnecting the host through the transceivers, and the $\overline{RD}$ and $\overline{WR}$ lines become outputs from the 82064 to allow access to the sector buffer. When the 82064 is done using the buffer, it deasserts $\overline{BCS}$ which again allows the host to access the local bus. The READ SECTOR command operates in a similar manner, except the buffer is loaded by the 82064 instead of the host.

Another control signal, BUFFER DATA REQUEST (BDRQ), can be used with a DMA controller to indicate that the 82064 is ready to send or receive data. When data transfer is via a programmed I/O environment, it is the responsibility of the host to interrogate the DRQ status bit to determine if the 82064 is ready (bit 3 of the status register). For further explanation, refer to the individual command descriptions and the A.C. Characteristics.

When INTRQ is asserted, the host is signaled that execution of a command has terminated (either a normal termination or an aborted command). For the READ SECTOR command, interrupts may be programmed to be asserted either at the termination of the command, or when BDRQ is asserted. INTRQ will remain active until the host reads the STATUS register to determine the cause of the termination, or writes a new command into the COMMAND register.

The 82064 asserts $\overline{\text{SDHLE}}$ whenever the SDH register is being written. This signal can be used to latch the drive and head select information in an external register for decoding. Figure 5 illustrates one method.

Drive Interface

The drive side of the 82064 WDC requires three sections of external logic. These are the control line buffer/receivers, data separator, and write precompensation. Figure 5 illustrates a drive interface.

The buffer/receivers condition the control lines to be driven down the cable to the drive. The control lines are typically single-ended, resistor terminated, TTL levels. The data lines to and from the drive also require buffering. This is typically done with differential RS-422 drivers. The interface specification for the drive will be found in the drive manufacturer's OEM

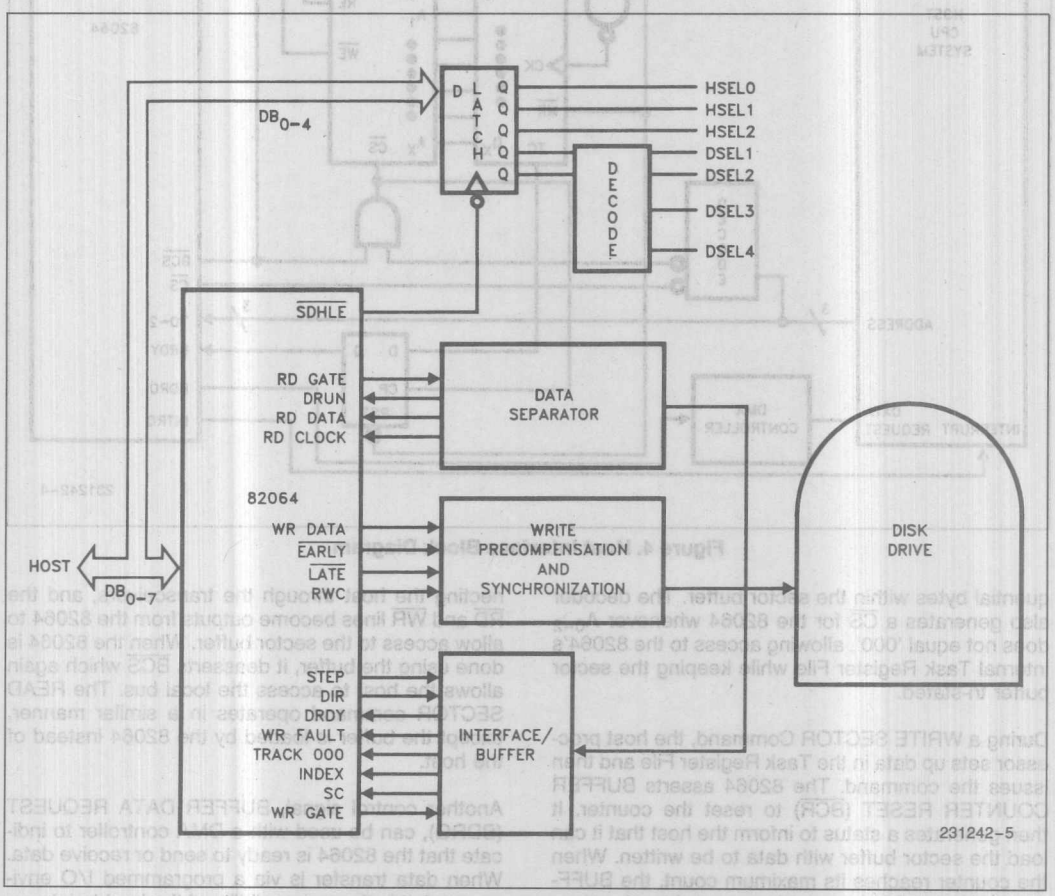


Figure 5. Drive Interface Block Diagram

manual. The 82064 supplies TTL compatible signals, and will interface to most buffer/driver devices.

The data recovery circuits consist of a phase locked loop, data separator, and associated components. The 82064 interacts with the data separator through the DATA RUN (DRUN) and RD GATE signals. A block diagram of a typical data separator circuit is shown in Figure 6. Read data from the drive is presented to the RD DATA input of the 82064, the reference multiplexor, and a retriggerable one shot. The RD GATE output will be deasserted when the 82064 is not inspecting data. The PLL should remain locked to the reference clock.

When any READ or WRITE command is initiated and a search for an address mark begins, the DRUN input is examined. The DRUN one-shot is set for slightly longer than one bit time, allowing it to retrigger constantly on a field of all ones or all zeroes. An internal counter times out to see that DRUN is asserted for two byte times. RD GATE is asserted by the 82064, switching the data separator to lock on to the incoming data stream. If DRUN is deasserted prior to an additional seven byte times, RD GATE is deasserted and the process is repeated. RD GATE will remain asserted until a non-zero, non-address mark byte is detected. The 82064 will then deassert RD GATE for two byte times to allow the PLL to lock back on the reference clock, and start the DRUN search again. If an address mark is detected, RD GATE remains asserted and the command will continue searching for the proper ID field. This sequence is shown in the flow chart in Figure 7.

The write precompensation circuitry is designed to reduce the drift in the data caused by interaction between bits. It is divided into two parts, REDUCED WRITE CURRENT (RWC) and $\overline{\text{EARLY}}$/$\overline{\text{LATE}}$ writing of bits. A block diagram of a typical write precompensation circuit is shown in Figure 8.

The cylinder in which the RWC line becomes active is controlled by the REDUCE WRITE CURRENT register in the Task Register File. When a cylinder is written which has a cylinder number greater than or equal to the contents of this register, the write current will be reduced. This will decrease the interaction between the bits.

Drift may also be caused by the bit pattern. With certain combinations of ones and zeroes some of the bits can drift far enough apart to be difficult to read without error. This phenomenon can be minimized by using $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ as described below. The 82064 examines three bits, the last one written, the one being written, and the next one to be written. From this, it determines whether to assert $\overline{\text{EARLY}}$ or $\overline{\text{LATE}}$. Since the bit leaving the 82064 has already been written, it is too late to make it early. Therefore, the external delay circuit must be as follows:

$\overline{\text{EARLY}}$ asserted and $\overline{\text{LATE}}$ deasserted = no delay

$\overline{\text{EARLY}}$ deasserted and $\overline{\text{LATE}}$ deasserted = one unit delay (typically 12–15 ns)

$\overline{\text{EARLY}}$ deasserted and $\overline{\text{LATE}}$ asserted = two units delay (typically 24–30 ns)

$\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are always active, and should be gated externally by the RWC signal. Figure 8 illustrates one method of using these signals.

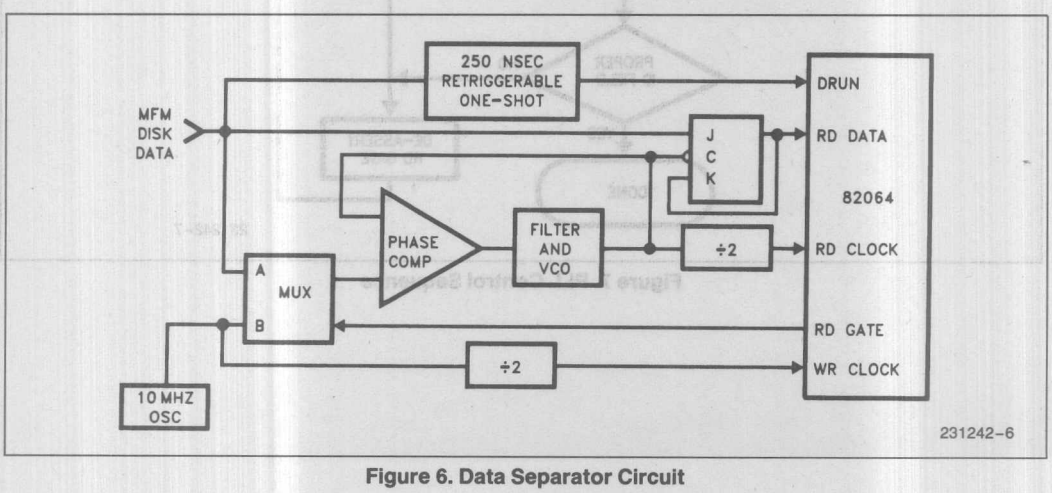


Figure 6. Data Separator Circuit

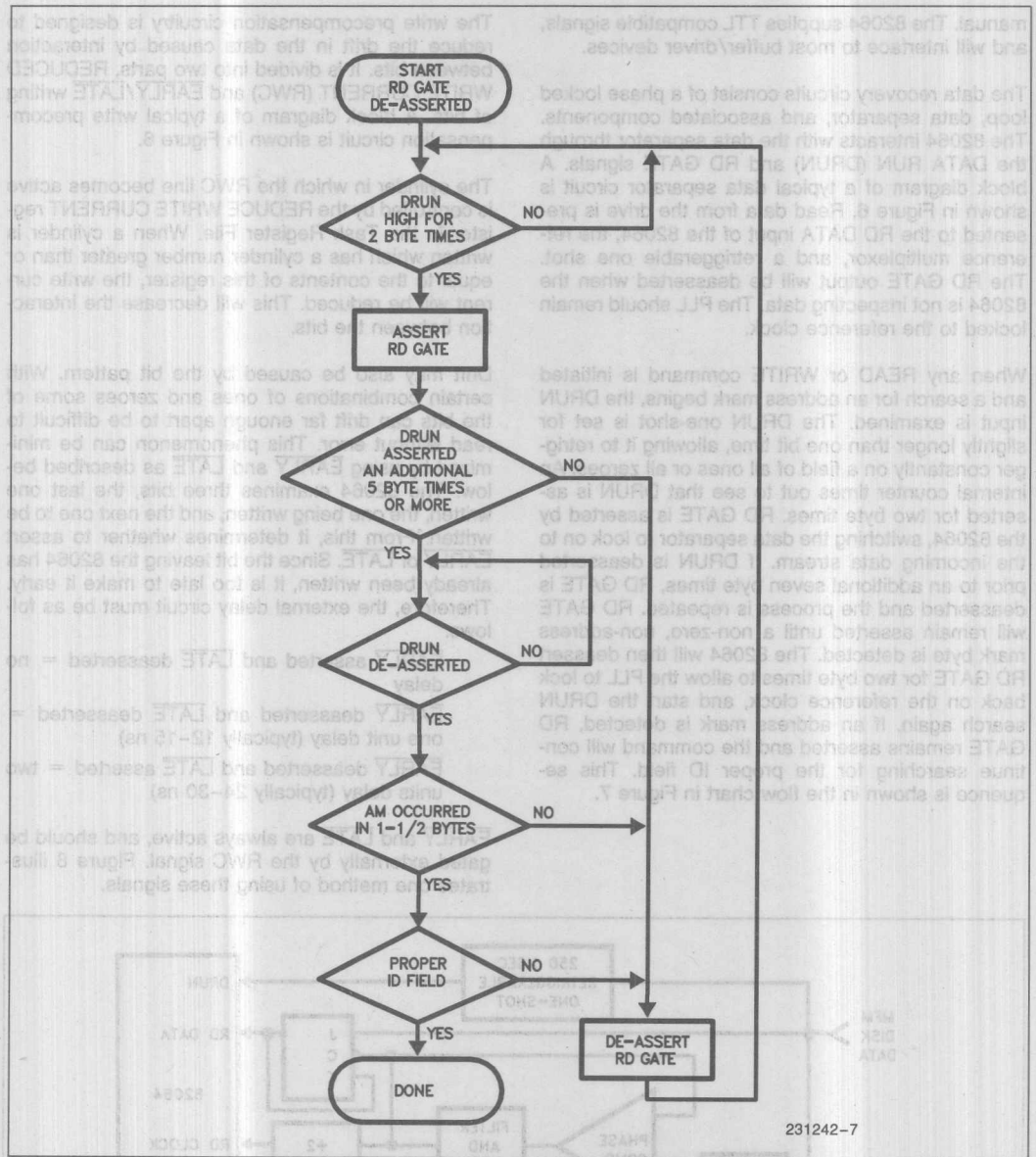


Figure 7. PLL Control Sequence

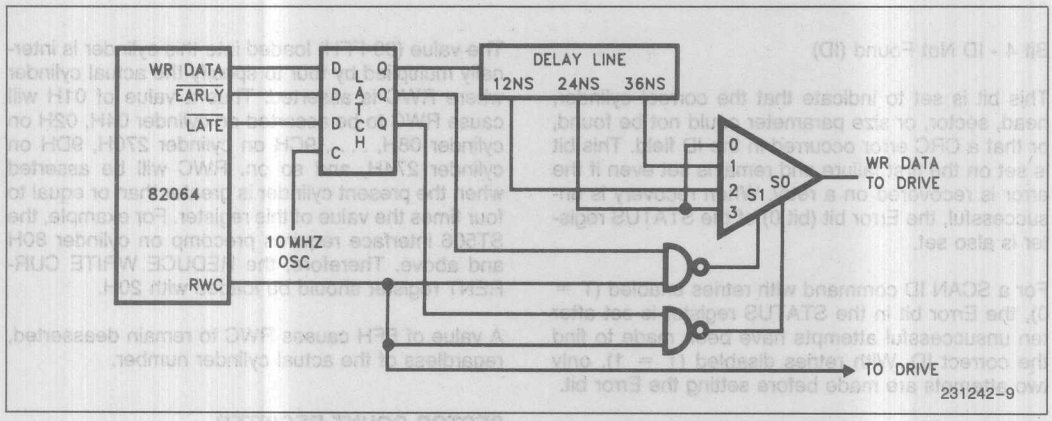


Figure 8. Write Precompensation Circuit

TASK REGISTER FILE

The Task Register File is a bank of nine registers used to hold parameter information pertaining to each command, status information, and the command itself. These registers and their addresses are:

A2	A1	A0	READ	WRITE
0	0	0	BUS TRI-STATED	BUS TRI-STATED
0	0	1	ERROR REGISTER	REDUCE WRITE CURRENT
0	1	0	SECTOR COUNT	SECTOR COUNT
0	1	1	SECTOR NUMBER	SECTOR NUMBER
1	0	0	CYLINDER LOW	CYLINDER LOW
1	0	1	CYLINDER HIGH	CYLINDER HIGH
1	1	0	SDH	SDH
1	1	1	STATUS	COMMAND

NOTE:
These registers are not cleared by $\overline{\text{RESET}}$ being asserted.

ERROR REGISTER

This read only register contains specific error information after the termination of a command. The bits are defined as follows:

7	6	5	4	3	2	1	0
BB	CRC/ECC	0	ID	0	AC	TK000	DAM

Bit 7 - Bad Block Detect (BB)

This bit is set when an ID field has been encountered that contains a bad block mark. It is used for bad sector mapping.

Bit 6 - CRC/ECC Data Field Error (CRC/ECC)

When in the CRC mode (SDH register, bit 7 = 0), this bit is set when a CRC error occurs in the data field. When retries are enabled, ten more attempts are made to read the sector correctly. If none of these attempts are successful bit 0 in the STATUS register is also set. If one of the attempts is successful, the CRC/ECC error bit remains set to inform the host that a marginal condition exists; however, bit 0 in the STATUS register is not set.

When in the ECC mode (SDH register, bit 7 = 1), this bit is set when the first non-zero syndrome is detected. When retries are enabled, up to ten attempts are made to correct the error. If the error is successfully corrected, this bit remains set; however, bit 2 of the STATUS register is also set to inform the host that the error has been corrected. If the error is not correctable, the CRC/ECC error bit remains set and bit 0 of the STATUS register is also set.

The data may be read even if uncorrectable errors exist.

NOTE: If the long mode (L) bit is set in the READ or WRITE command, no error checking is performed.

Bit 5 - Reserved

Not used. Forced to zero.

Bit 4 - ID Not Found (ID)

This bit is set to indicate that the correct cylinder, head, sector, or size parameter could not be found, or that a CRC error occurred in the ID field. This bit is set on the first failure and remains set even if the error is recovered on a retry. When recovery is unsuccessful, the Error bit (bit 0) of the STATUS register is also set.

For a SCAN ID command with retries enabled (T = 0), the Error bit in the STATUS register is set after ten unsuccessful attempts have been made to find the correct ID. With retries disabled (T = 1), only two attempts are made before setting the Error bit.

For a READ or WRITE command with retries enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan and auto-seek are performed. Then ten more retries are made before setting the Error bit. When retries are disabled (T = 1), only two tries are made. No auto-scan or auto-seek operations are performed.

Bit 3 - Reserved

Not used. Forced to zero.

Bit 2 - Aborted Command (AC)

Command execution is aborted and this bit is set if a command was issued while DRDY is deasserted or WR FAULT is asserted. This bit will also be set if an undefined command is written to the COMMAND register; however, an implied seek will be executed.

Bit 1 - Track 000 Error (TK000)

This bit is set during the execution of a RESTORE command if the TRACK 000 pin has not gone active after the issuance of 2047 step pulses.

Bit 0 - Data Address Mark (DAM) Not Found

This bit is set during the execution of a READ SECTOR command if the DAM is not found following the proper sector ID.

REDUCE WRITE CURRENT REGISTER

This register is used to define the cylinder number where the RWC output (Pin 33) is asserted.

7	6	5	4	3	2	1	0
CYLINDER NUMBER ÷ 4							

The value (00-FFH) loaded into this cylinder is internally multiplied by four to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to be asserted on cylinder 04H, 02H on cylinder 08H, . . . , 9CH on cylinder 270H, 9DH on cylinder 274H, and so on. RWC will be asserted when the present cylinder is greater than or equal to four times the value of this register. For example, the ST506 interface requires precomp on cylinder 80H and above. Therefore, the REDUCE WRITE CURRENT register should be loaded with 20H.

A value of FFH causes RWC to remain deasserted, regardless of the actual cylinder number.

SECTOR COUNT REGISTER

This register is used to define the number of sectors that need to be transferred to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command.

7	6	5	4	3	2	1	0
NUMBER OF SECTORS							

The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a one sector transfer, etc. This register is a "don't care" when single sector commands are specified.

SECTOR NUMBER REGISTER

This register holds the sector number of the desired sector.

7	6	5	4	3	2	1	0
SECTOR NUMBER							

For a multiple sector command, it specifies the first sector to be transferred. It is incremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formating a disk. See the WRITE FORMAT command description for further explanation.

CYLINDER NUMBER LOW REGISTER

This register holds the lower byte of the desired cylinder number.

7	6	5	4	3	2	1	0
LS BYTE OF CYL. NUMBER							

It is used with the CYLINDER NUMBER HIGH register to specify the desired cylinder number over a range of 0 to 2047.

CYLINDER NUMBER HIGH REGISTER

This register holds the three most significant bits of the desired cylinder number.

7	6	5	4	3	2	1	0
x	x	x	x	x	#	#	#

The CYLINDER NUMBER LOW/HIGH register pair determine where the R/W heads are to be positioned. The host writes the desired cylinder number into these registers. Internal to the 82064 is another pair of registers that hold the present head location. When any command other than a RESTORE is executed, the internal head location registers are compared to the CYLINDER NUMBER registers to determine how many cylinders to move the heads and in what direction.

The internal head location registers are updated to equal the CYLINDER NUMBER registers after the completion of the seek.

When a RESTORE command is executed, the internal head location registers are reset to zero while DIR and STEP move the heads to track zero.

SECTOR/DRIVE/HEAD (SDH) REGISTER

The SDH register contains the desired sector size, drive number, and head parameters. The format is shown in Figure 9. The EXT bit (bit 7) is used to select between the CRC or ECC mode. When bit 7 = 1 the ECC mode is selected for the data field. When bit 7 = 0 the CRC mode is selected.

The SDH byte written in the ID field of the disk by the FORMAT command is different than the SDH register contents. The recorded SDH byte does not have the drive number recorded, but does have the bad block mark written. The format of the SDH byte written on the disk is:

7	6	5	4	3	2	1	0
BAD B.	SIZE		0	0	HEAD		

STATUS REGISTER

The status register is used to inform the host of certain events performed by the 82064, as well as reporting status from the drive control lines. Reading the STATUS register deasserts INTRQ. The format is:

7	6	5	4	3	2	1	0
BUSY	READY	WF	SC	DRQ	DWC	CIP	ERR

Bit 7 - Busy

This bit is asserted when a command is written into the COMMAND register and, except for the READ command, is deasserted at the end of the command. When executing a READ command, Busy will be deasserted when the sector buffer is full. Commands should not be loaded into the COMMAND register when Busy is set. When the Busy bit is set, no other bits in the STATUS or ERROR registers are valid.

Bit 6 - Ready

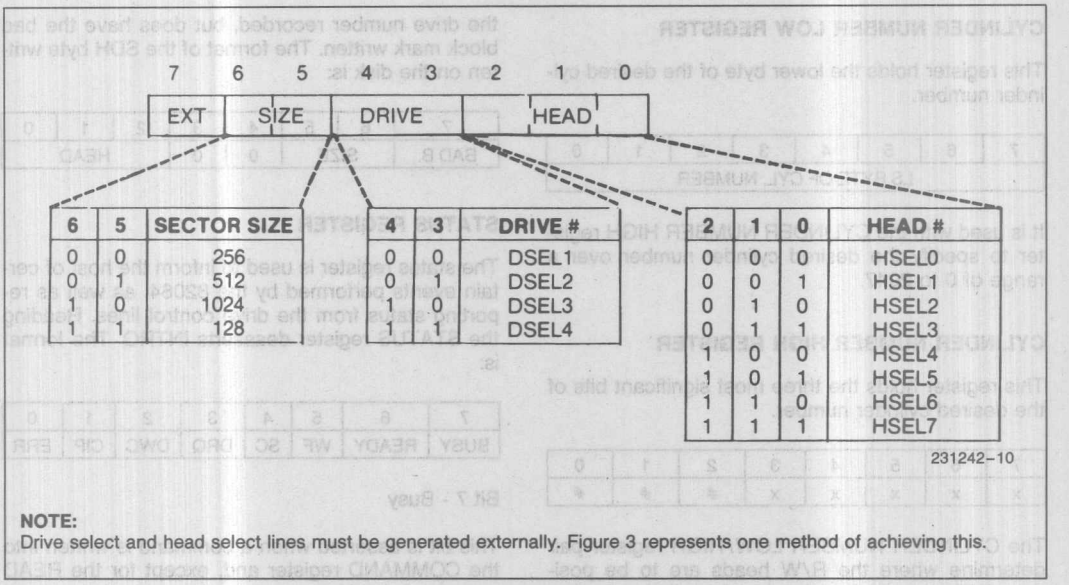
This bit reflects the status of DRDY (pin 28). When this bit equals zero, the command is aborted and the status of this bit is latched.

Bit 5 - Write Fault (WF)

This bit reflects the status of WR FAULT (pin 30). When this bit equals one the command is aborted, INTRQ is asserted, and the status of this bit is latched.

Bit 4 - Seek Complete (SC)

This bit reflects the status of SC (pin 32). When a seek or implied seek has been initiated by a command, execution of the command pauses until the seek is complete. This bit is latched after an aborted command error.



NOTE:
Drive select and head select lines must be generated externally. Figure 3 represents one method of achieving this.

Figure 9. SDH Register Format

Bit 3 - Data Request (DRQ)

The DRQ bit reflects the status of BDRQ (pin 36). It is asserted when the sector buffer must be written into or read from. DRQ and BDRQ remain asserted until BRDY indicates that the sector buffer has been filled or emptied, depending upon the command. BDRQ can be used for DMA interfacing, while DRQ is used in a programmed I/O environment.

Bit 2 - Data Was Corrected (DWC)

When set, this bit indicates that an ECC error has been detected during a read operation, and that the data in the sector buffer has been corrected. This provides the user with an indication that there may be a marginal condition within the drive before the errors become uncorrectable. This bit is forced to zero when not in the ECC mode.

Bit 1 - Command In Progress (CIP)

When this bit is set a command is being executed and a new command should not be loaded. Although a command is being executed, the sector buffer is still available for access by the host. When the 82064 is no longer Busy (bit 7 = 0) the STATUS register can be read. If other registers are read while CIP is set the contents of the STATUS register will be returned.

Bit 0 - Error

This bit is set whenever any bits in the ERROR register are set. It is the logical 'or' of the bits in the ERROR register and may be used by the host processor to quickly check for nonrecoverable errors. The host must read the ERROR register to determine what type of error occurred. This bit is reset when a new command is written into the COMMAND register.

COMMAND REGISTER

The command to be executed is written into this write-only register:

7	6	5	4	3	2	1	0
COMMAND							

The command sets Busy and CIP, and begins to execute as soon as it is written into this register. Therefore, all necessary information should be loaded into the Task Register File prior to entering the command. Any attempt to write a register will be ignored until command execution has terminated, as indicated by the CIP bit being cleared. INTRQ is deasserted when the COMMAND register is written.

COMMAND	7	6	5	4	3	2	1	0
RESTORE	0	0	0	1	R3	R2	R1	R0
SEEK	0	1	1	1	R3	R2	R1	R0
READ SECTOR	0	0	1	0	I	M	L	T
WRITE SECTOR	0	0	1	1	0	M	L	T
SCAN ID	0	1	0	0	0	0	0	T
WRITE FORMAT	0	1	0	1	0	G	0	0
COMPUTE CORRECTION	0	0	0	0	1	0	0	0
SET PARAMETER	0	0	0	0	0	0	0	S

R_{3-0} = Stepping Rate Field

For 5 MHz WR CLOCK:

R_{3-0} = 0000 35 μs
0001 0.5 ms
0010 1.0 ms
0011 1.5 ms
0100 2.0 ms
0101 2.5 ms
0110 3.0 ms
0111 3.5 ms
1000 4.0 ms
1001 4.5 ms
1010 5.0 ms
1011 5.5 ms
1100 6.0 ms
1101 6.5 ms
1110 3.2 μs
1111 16 μs

I = Interrupt Control

I = 0 INTRQ occurs with BDRQ/DRQ indicating the sector buffer is full. Valid only when M = 0.

I = 1 INTRQ occurs when the command is completed and the host has read the sector buffer.

M = Multiple Sector Flag

M = 0 Transfer one sector. Ignore the SECTOR COUNT register.

M = 1 Transfer multiple sectors.

L = Long Mode

L = 0 Normal mode. Normal CRC or ECC functions are performed.

L = 1 Long mode. No CRC or ECC bytes are developed or error checking performed on the data field. The 82064 appends the four additional bytes supplied by the host or disk to the data field.

T = Retry Enable

T = 0 Enable retries.

T = 1 Disable retries.

G = Gap Filler Byte.

G = 0. Gaps 1, 3 and pad bytes "4E".

G = 1. Gaps 1, 3 and pad bytes "AA".

S = Error Correction Span

S = 0 5-bit span.

S = 1 11-bit span.

RESTORE COMMAND

The RESTORE command is used to position the R/W heads over track zero. It is usually issued by the host when a drive has just been turned on. The 82064 forces an auto-restore when a FORMAT command has been issued following a drive number change.

The actual step rate used for the RESTORE command is determined by the seek complete time. A step pulse is issued and the 82064 waits for a rising edge on the SC line before issuing the next pulse. If the rising edge of SC has not occurred within ten revolutions (INDEX pulses) the 82064 switches to sensing the level of SC. If after 2047 step pulses the TRACK 000 line does not go active the 82064 will set the TRACK 000 bit in the ERROR register, assert INTRQ, and terminate execution of the command. An interrupt will also occur if WR FAULT is asserted on DRDY is deasserted at any time during execution.

The rate field specified (R_{3-0}) is stored in an internal register for future use in commands with implied seeks.

A flowchart of the RESTORE command is shown in Figure 10.

SEEK COMMAND

The SEEK command can be used for overlapping seeks on multiple drives. The step rate used is taken from the Rate Field of the command, and is stored in an internal register for future use by those commands with implied seek capability.

The direction and number of step pulses needed are calculated by comparing the contents of the CYLINDER NUMBER registers in the Task Register File to the present cylinder position stored internally. After all the step pulses have been issued the present cylinder position is updated, INTRQ is asserted, and the command terminated.

If DRDY is deasserted or WR FAULT is asserted during the execution of the command, INTRQ is asserted and the command aborts setting the AC bit in the ERROR register.

If an implied seek is performed, the step rate indicated by the rate field is used for all but the last step pulse. On the last pulse, the command execution continues until the rising edge of SC is detected. If 10 INDEX pulses are received without a rising edge of SC, the 82064 will switch to sensing the level of SC.

A flowchart of the SEEK command flow is shown in Figure 11.

READ SECTOR

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the command, the 82064 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation takes place, and a seek is initiated. As stated in the description of the SEEK command, if an implied seek occurs, the step rate specified by the rate field is used for all but the last step pulse. On the last step pulse the seek continues until the rising edge of SC is detected.

If the 82064 detects a change in the drive number since the last command, an auto-scan ID is performed. This updates the internal cylinder position register to reflect the current drive before the seek begins.

After the 82064 senses SC (with or without an implied seek) it must find an ID field with the correct cylinder number, head, sector size, and CRC. If retries are enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan ID and auto-seek are performed. Then ten more retries are attempted before setting the ID Not Found error bit. When retries are disabled (T = 1) only two tries are made. No auto-scan or auto-seek operations are performed.

When the data address mark (DAM) is found, the 82064 is ready to transfer data into the sector buffer. When the disk has filled the sector buffer, the 82064 asserts BDRQ and DRQ and then checks the I flag. If I = 0, INTRQ is asserted, signaling the host to read the contents of the sector buffer. If I = 1, INTRQ occurs after the host has read the sector buffer and the command has terminated. If after successfully reading the ID field, the DAM is not found the DAM Not Found bit in the ERROR register is set.

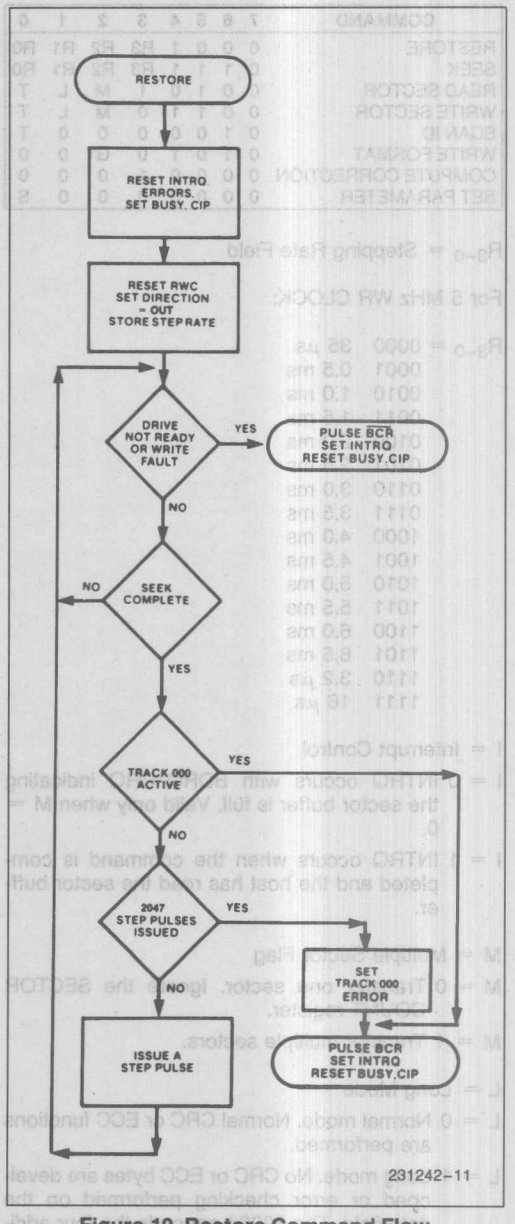


Figure 10. Restore Command Flow

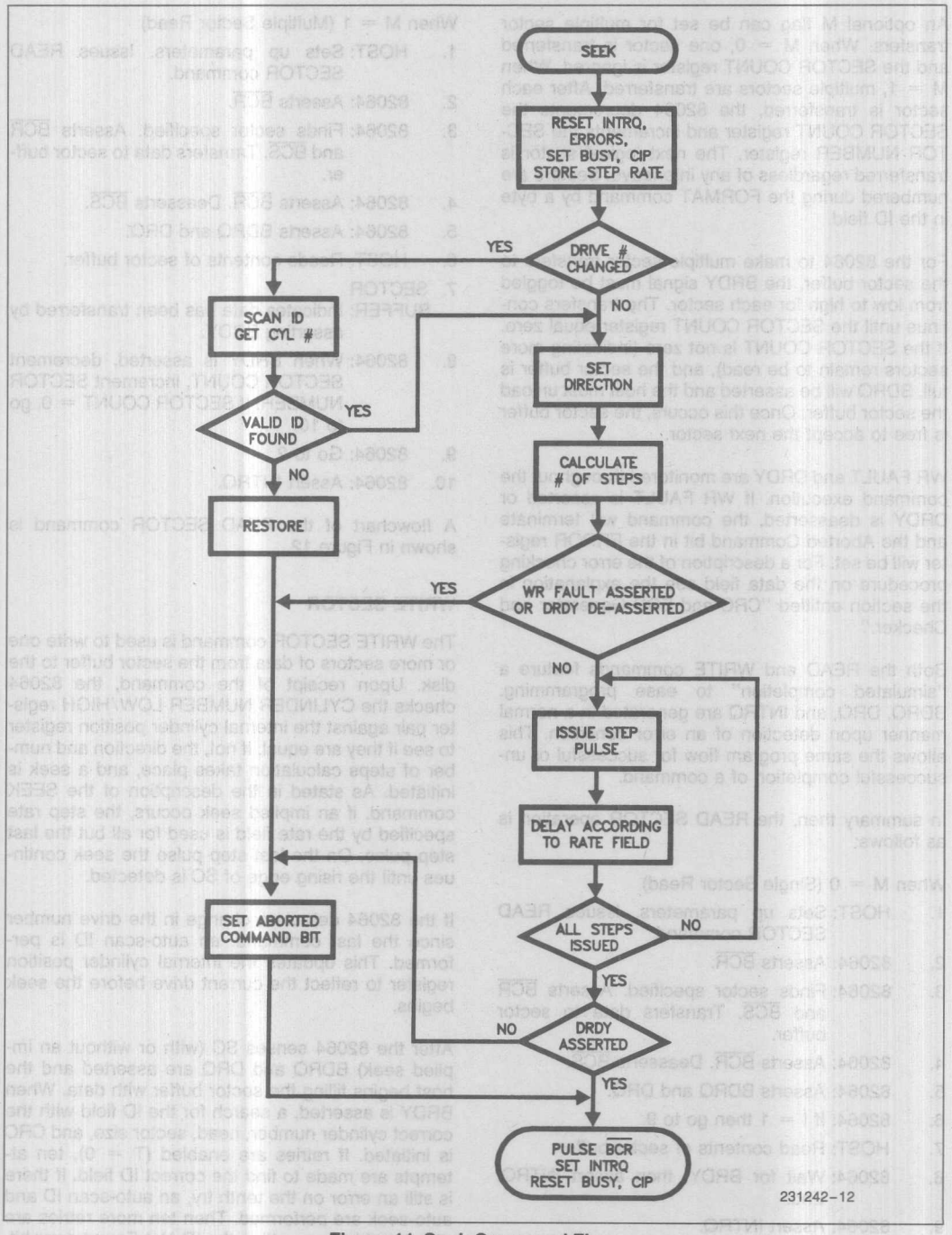


Figure 11. Seek Command Flow

An optional M flag can be set for multiple sector transfers. When M = 0, one sector is transferred and the SECTOR COUNT register is ignored. When M = 1, multiple sectors are transferred. After each sector is transferred, the 82064 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector is transferred regardless of any interleave. Sectors are numbered during the FORMAT command by a byte in the ID field.

For the 82064 to make multiple sector transfers to the sector buffer, the BRDY signal must be toggled from low to high for each sector. The transfers continue until the SECTOR COUNT register equal zero. If the SECTOR COUNT is not zero (indicating more sectors remain to be read), and the sector buffer is full, BDRQ will be asserted and the host must unload the sector buffer. Once this occurs, the sector buffer is free to accept the next sector.

WR FAULT and DRDY are monitored throughout the command execution. If WR FAULT is asserted or DRDY is deasserted, the command will terminate and the Aborted Command bit in the ERROR register will be set. For a description of the error checking procedure on the data field see the explanation in the section entitled "CRC and ECC Generator and Checker."

Both the READ and WRITE commands feature a "simulated completion" to ease programming. BDRQ, DRQ, and INTRQ are generated in a normal manner upon detection of an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

In summary then, the READ SECTOR operation is as follows:

When M = 0 (Single Sector Read)

1. HOST: Sets up parameters. Issues READ SECTOR command.
2. 82064: Asserts $\overline{BCR}$.
3. 82064: Finds sector specified. Asserts $\overline{BCR}$ and $\overline{BCS}$. Transfers data to sector buffer.
4. 82064: Asserts $\overline{BCR}$. Deasserts $\overline{BCS}$.
5. 82064: Asserts BDRQ and DRQ.
6. 82064: If I = 1 then go to 9.
7. HOST: Read contents of sector buffer.
8. 82064: Wait for BRDY, then assert INTRQ. End.
9. 82064: Assert INTRQ.
10. HOST: Read contents of sector buffer. End.

When M = 1 (Multiple Sector Read)

1. HOST: Sets up parameters. Issues READ SECTOR command.
2. 82064: Asserts $\overline{BCR}$.
3. 82064: Finds sector specified. Asserts $\overline{BCR}$ and $\overline{BCS}$. Transfers data to sector buffer.
4. 82064: Asserts $\overline{BCR}$. Deasserts $\overline{BCS}$.
5. 82064: Asserts BDRQ and DRQ.
6. HOST: Reads contents of sector buffer.
7. SECTOR BUFFER: Indicates data has been transferred by asserting BRDY.
8. 82064: When BRDY is asserted, decrement SECTOR COUNT, increment SECTOR NUMBER. If SECTOR COUNT = 0, go to 10.
9. 82064: Go to 2.
10. 82064: Assert INTRQ.

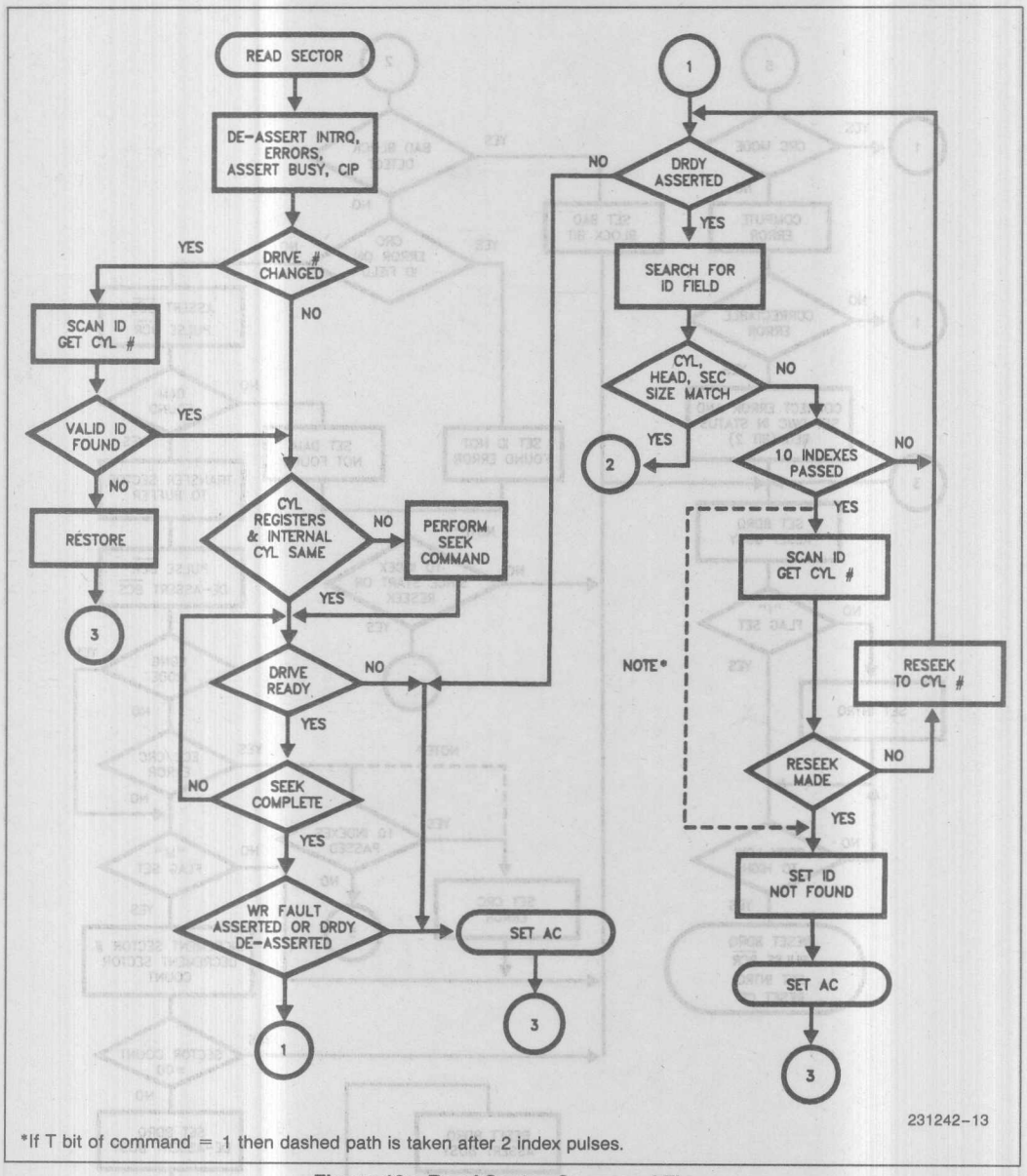
A flowchart of the READ SECTOR command is shown in Figure 12.

WRITE SECTOR

The WRITE SECTOR command is used to write one or more sectors of data from the sector buffer to the disk. Upon receipt of the command, the 82064 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation takes place, and a seek is initiated. As stated in the description of the SEEK command, if an implied seek occurs, the step rate specified by the rate field is used for all but the last step pulse. On the last step pulse the seek continues until the rising edge of SC is detected.

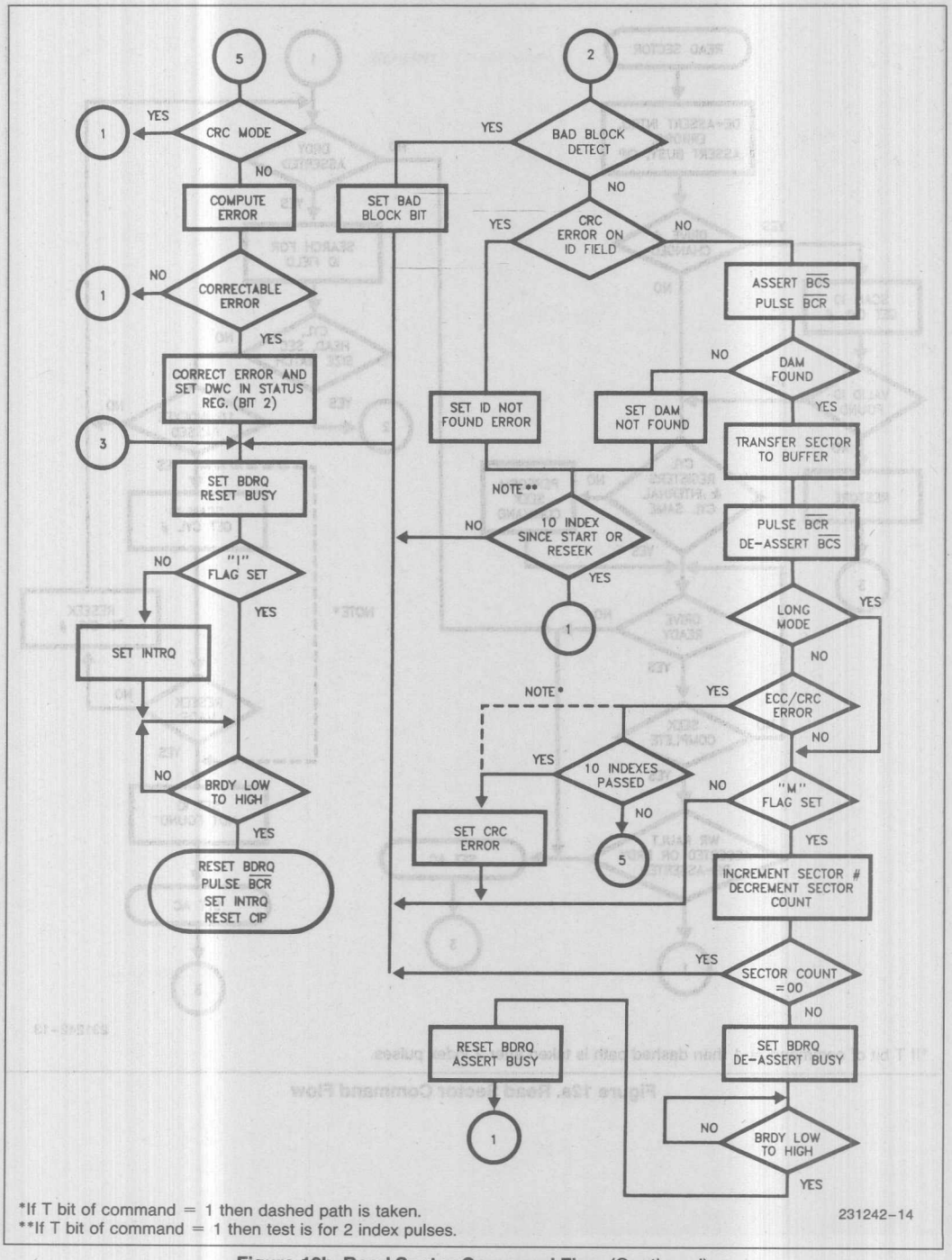
If the 82064 detects a change in the drive number since the last command, an auto-scan ID is performed. This updates the internal cylinder position register to reflect the current drive before the seek begins.

After the 82064 senses SC (with or without an implied seek) BDRQ and DRQ are asserted and the host begins filling the sector buffer with data. When BRDY is asserted, a search for the ID field with the correct cylinder number, head, sector size, and CRC is initiated. If retries are enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan ID and auto-seek are performed. Then ten more retries are attempted before setting the ID Not Found error bit. When retries are disabled (T = 1) only two tries are made. No auto-scan or auto-seek operations are performed.



*If T bit of command = 1 then dashed path is taken after 2 index pulses.

Figure 12a. Read Sector Command Flow



*If T bit of command = 1 then dashed path is taken.
**If T bit of command = 1 then test is for 2 index pulses.

231242-14

Figure 12b. Read Sector Command Flow (Continued)

When the correct ID is found, WR GATE is asserted and data is written to the disk. When the CRC/ECC bit (SDH Register, bit 7) is zero, the 82064 generates a two byte CRC character to be appended to the data. When the CRC/ECC bit is one, four ECC bytes replace the CRC character. When L = 1, the polynomial generator is inhibited and neither CRC or ECC bytes are generated. Instead four bytes of data supplied by the host are written.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is incremented and the SECTOR COUNT register is decremented. If BRDY is asserted after the first sector is read from the sector buffer, the 82064 continues to read data from the sector buffer for the next sector. If BRDY is deasserted, the 82064 asserts BDRQ and waits for the host to place more data in the sector buffer.

In summary then, the WRITE SECTOR operation is as follows:

When M = 0,1

1. HOST: Sets up parameters. Issues WRITE SECTOR command.
2. 82064: Asserts BDRQ and DRQ.
3. HOST: Loads sector buffer with data.
4. 82064: Waits for rising edge of BRDY.
5. 82064: Finds specified ID field. Writes sector to disk.
6. 82064: If M = 0, asserts INTRQ. End.
7. 82064: Increments SECTOR NUMBER. Decrements SECTOR COUNT.
8. 82064: IF SECTOR COUNT = 0, assert INTRQ. End.
9. 82064: Go to 2.

A flowchart of the WRITE SECTOR command is shown in Figure 13.

SCAN ID

The SCAN ID command is used to update the SDH, SECTOR NUMBER, and CYLINDER NUMBER LOW/HIGH registers.

After the command is loaded, the SC line is sampled until it is valid. The DRDY and WR FAULT lines are also monitored throughout execution of the command. If a fault occurs the command is aborted and the appropriate error bits are set. When the first ID field is found, the ID information is loaded into the SDH, SECTOR NUMBER, and CYLINDER NUMBER registers. The internal cylinder position register is also updated. If this is an auto-scan caused by a change in drive numbers, only the internal position register is updated. If a bad block is detected, the BAD BLOCK bit will also be set.

If an ID field is not found, or if a CRC error occurs, and if retries are enabled (T = 0), ten attempts are made to read it. If retries are disabled (T = 1), only two tries are made. There is no auto-seek in this command and the sector buffer is not disturbed.

A flowchart of the SCAN ID command is shown in Figure 14.

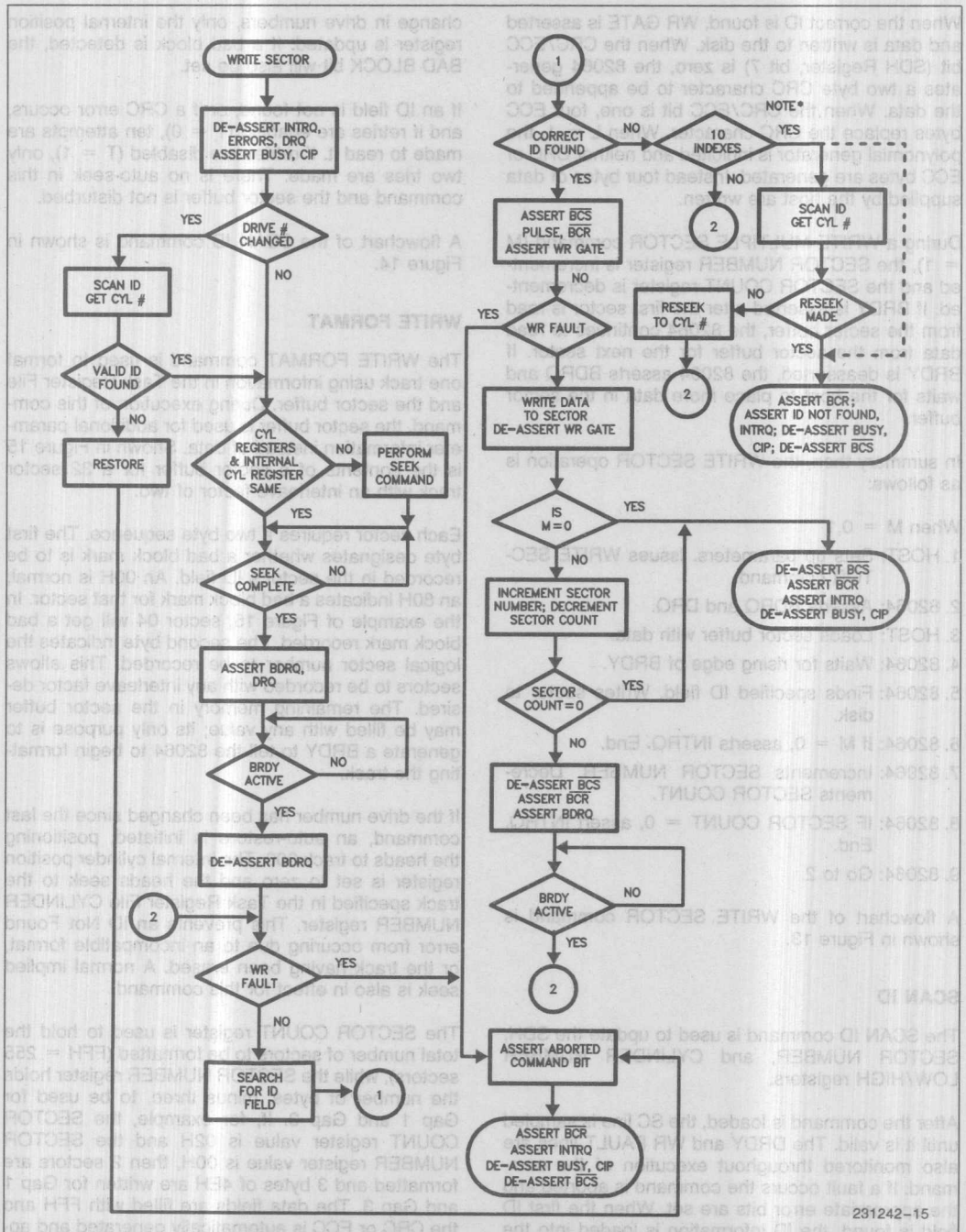
WRITE FORMAT

The WRITE FORMAT command is used to format one track using information in the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of data. Shown in Figure 15 is the contents of a sector buffer for a 32 sector track with an interleave factor of two.

Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. An 00H is normal; an 80H indicates a bad block mark for that sector. In the example of Figure 15, sector 04 will get a bad block mark recorded. The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may be filled with any value; its only purpose is to generate a BRDY to tell the 82064 to begin formatting the track.

If the drive number has been changed since the last command, an auto-restore is initiated, positioning the heads to track 000. The internal cylinder position register is set to zero and the heads seek to the track specified in the Task Register File CYLINDER NUMBER register. This prevents an ID Not Found error from occuring due to an incompatible format, or the track having been erased. A normal implied seek is also in effect for this command.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (FFH = 255 sectors), while the SECTOR NUMBER register holds the number of bytes, minus three, to be used for Gap 1 and Gap 3. If, for example, the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are formatted and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC or ECC is automatically generated and appended. After the last sector is written the track is filled with 4EH.



*If retries disabled then dashed path is taken after 2 index pulses.

Figure 13. Write Sector Command Flow

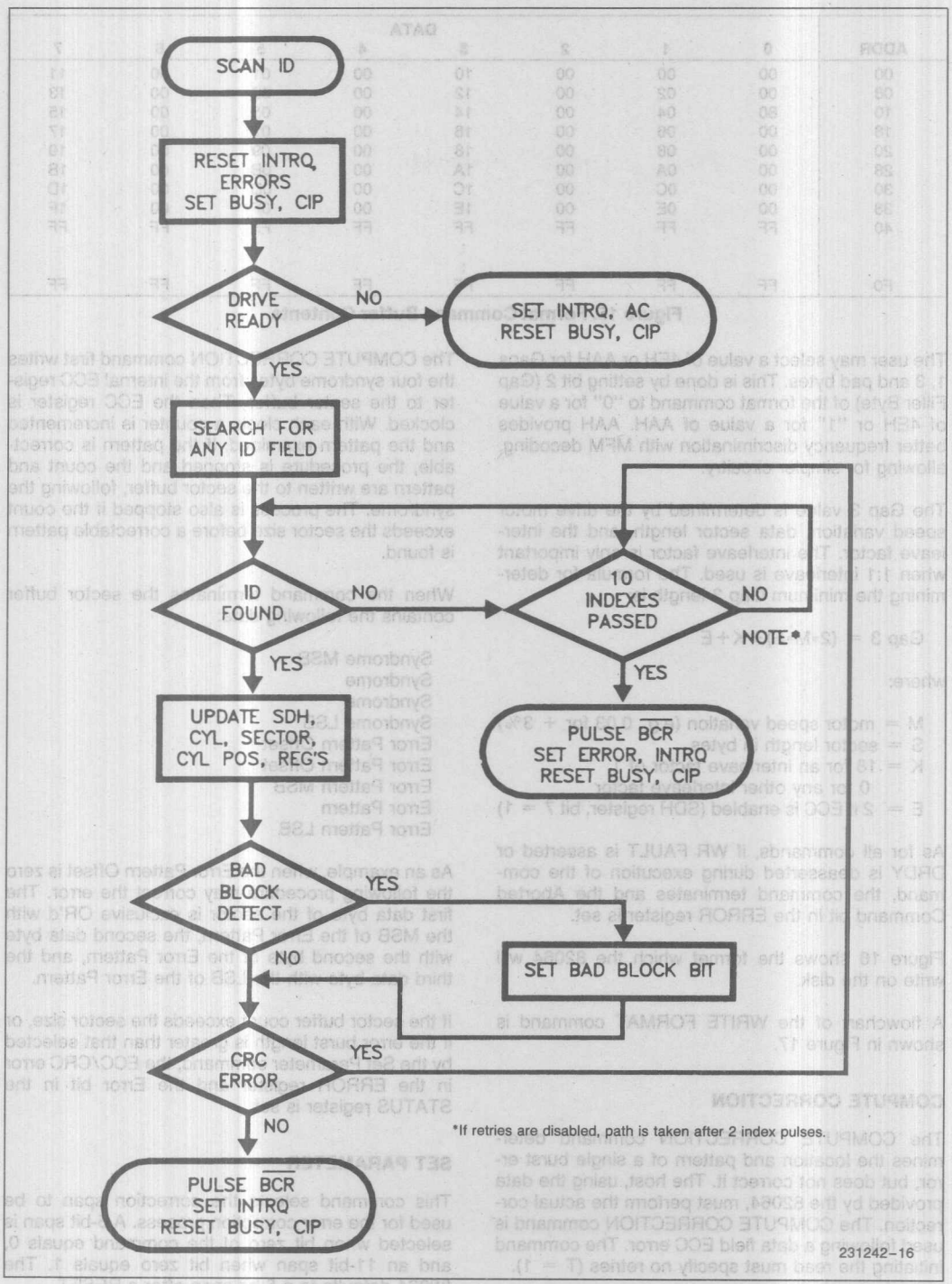


Figure 14. Scan ID Command Flow

ADDR	DATA 0	1	2	3	4	5	6	7
00	00	00	00	10	00	01	00	11
08	00	02	00	12	00	03	00	13
10	80	04	00	14	00	05	00	15
18	00	06	00	16	00	07	00	17
20	00	08	00	18	00	09	00	19
28	00	0A	00	1A	00	0B	00	1B
30	00	0C	00	1C	00	0D	00	1D
38	00	0E	00	1E	00	0F	00	1F
40	FF	FF	FF	FF	FF	FF	FF	FF
				:				
				:				
F0	FF	FF	FF	FF	FF	FF	FF	FF

Figure 15. Format Command Buffer Contents

The user may select a value of 4EH or AAH for Gaps 1, 3 and pad bytes. This is done by setting bit 2 (Gap Filler Byte) of the format command to "0" for a value of 4EH or "1" for a value of AAH. AAH provides better frequency discrimination with MFM decoding, allowing for simpler circuitry.

The Gap 3 value is determined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 interleave is used. The formula for determining the minimum Gap 3 length is:

$$\text{Gap 3} = (2*M*S)+K+E$$

where:

M = motor speed variation (e.g., 0.03 for + 3%)
S = sector length in bytes
K = 18 for an interleave factor of 1
    0 for any other interleave factor
E = 2 if ECC is enabled (SDH register, bit 7 = 1)

As for all commands, if WR FAULT is asserted or DRDY is deasserted during execution of the command, the command terminates and the Aborted Command bit in the ERROR register is set.

Figure 16 shows the format which the 82064 will write on the disk.

A flowchart of the WRITE FORMAT command is shown in Figure 17.

COMPUTE CORRECTION

The COMPUTE CORRECTION command determines the location and pattern of a single burst error, but does not correct it. The host, using the data provided by the 82064, must perform the actual correction. The COMPUTE CORRECTION command is used following a data field ECC error. The command initiating the read must specify no retries (T = 1).

The COMPUTE CORRECTION command first writes the four syndrome bytes from the internal ECC register to the sector buffer. Then the ECC register is clocked. With each clock, a counter is incremented and the pattern examined. If the pattern is correctable, the procedure is stopped and the count and pattern are written to the sector buffer, following the syndrome. The process is also stopped if the count exceeds the sector size before a correctable pattern is found.

When the command terminates the sector buffer contains the following data:

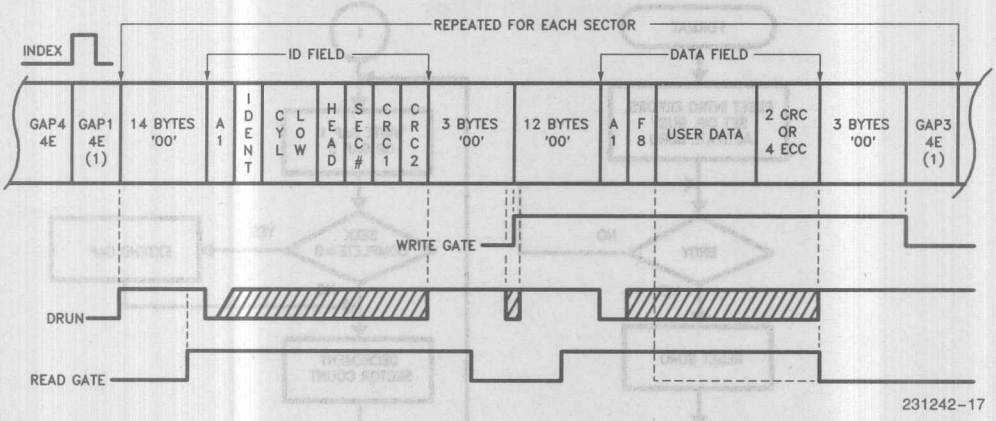
Syndrome MSB
Syndrome
Syndrome
Syndrome LSB
Error Pattern Offset
Error Pattern Offset
Error Pattern MSB
Error Pattern
Error Pattern LSB

As an example, when the Error Pattern Offset is zero the following procedure may correct the error. The first data byte of the sector is exclusive OR'd with the MSB of the Error Pattern, the second data byte with the second byte of the Error Pattern, and the third data byte with the LSB of the Error Pattern.

If the sector buffer count exceeds the sector size, or if the error burst length is greater than that selected by the Set Parameter command, the ECC/CRC error in the ERROR register and the Error bit in the STATUS register is set.

SET PARAMETER

This command selects the correction span to be used for the error correction process. A 5-bit span is selected when bit zero of the command equals 0, and an 11-bit span when bit zero equals 1. The 82064 defaults to a 5-bit span after a RESET.



ID FIELD		
A1	=	A1H with 0AH Clock
IDENT	=	Bits 3, 1, 0 = Cylinder High FE = 0-255 Cylinders FF = 256-511 Cylinders FC = 512-767 Cylinders FD = 768-1023 Cylinders F6 = 1024-1279 Cylinders F7 = 1280-1535 Cylinders F4 = 1536-1791 Cylinders F5 = 1792-2047 Cylinders
HEAD	=	Bits 0, 1, 2 = Head Number Bits 3, 4 = 0 Bits 5, 6 = Sector Size Bit 7 = Bad Block Mark
Sec #	=	Logical Sector Number
DATA FIELD		
A1	=	A1H with 0AH clock
F8	=	Data Address Mark; Normal Clock
USER	=	Data Field 128 to 1024 Bytes

NOTE:
1. GAP 1 and 3 length determined by Sector Number Register contents during formatting.

Figure 16. Track Format

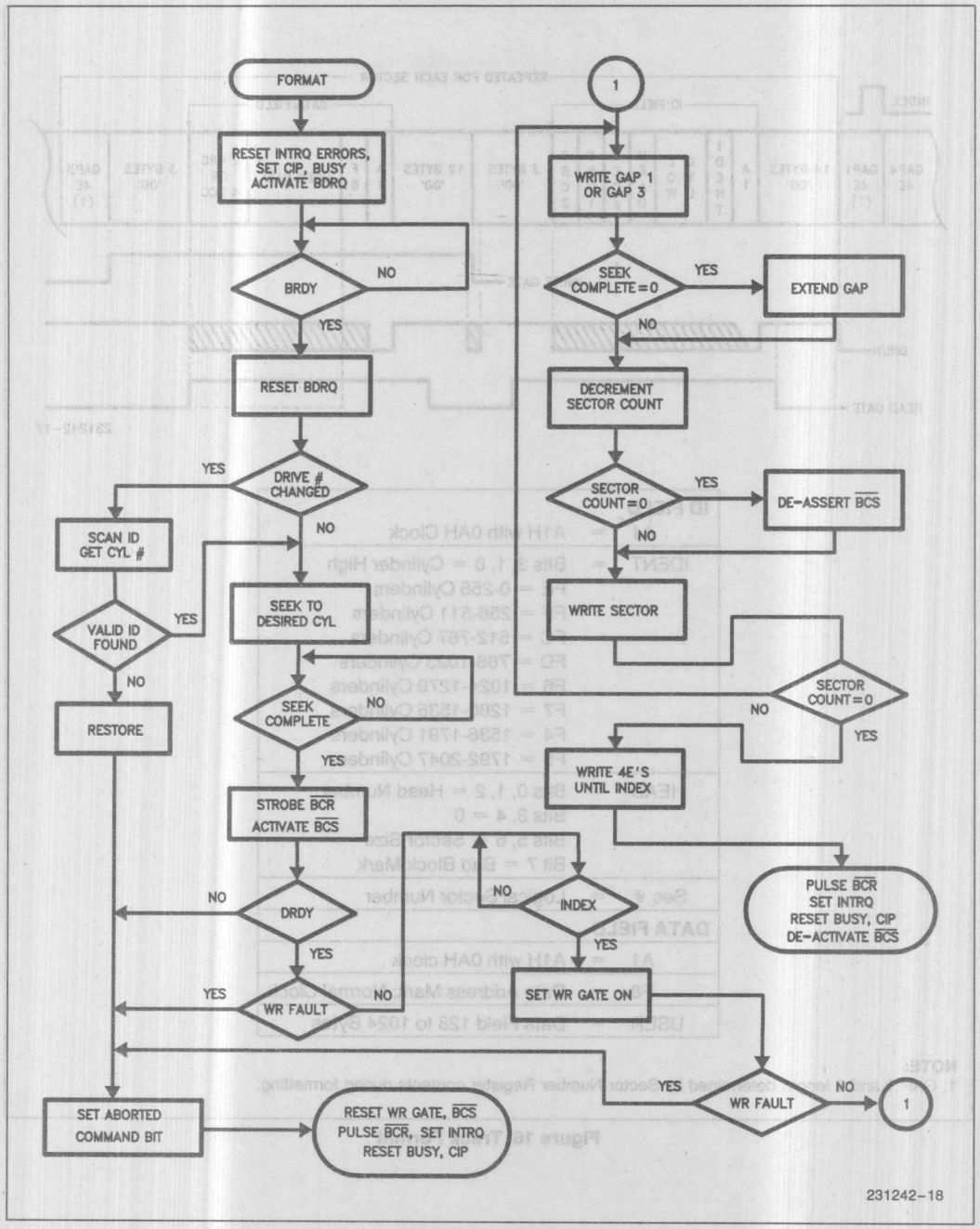


Figure 17. Write Format Command Flow

ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Supply Voltage .....−0.5V to +8V
Voltage on Any Input .....GND − 2V to +6.5V
Voltage on Any Output .GND − 0.5V to V_{CC} + 0.5V
Power Dissipation .....1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

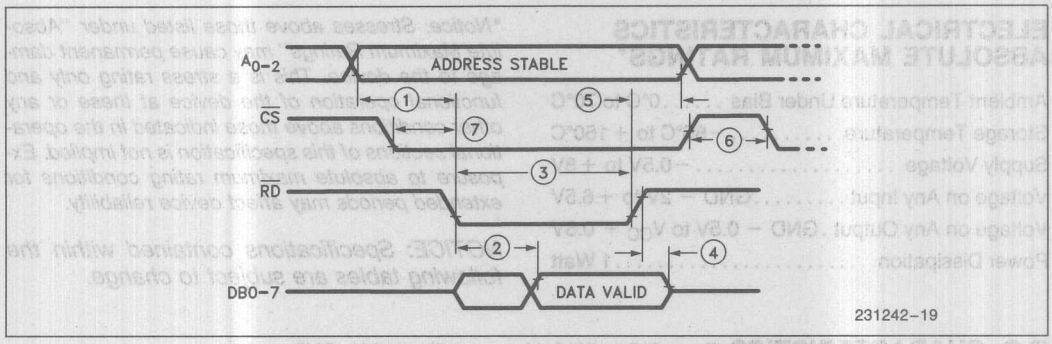
D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = +5V ±10%; GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{IL}	Input Leakage Current		±10	µA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Leakage Current		±10	µA	V_{OUT} = V_{CC} to 0.45V
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{OH}	Output High Voltage	V_{CC} − 0.4 3.0		V	I_{OH} = −100 µA I_{OH} = −2.5 mA
V_{OL}	Output Low Voltage		0.4 0.45	V	I_{OL} = 2.5 mA 6.0 mA P21, 22, 23
I_{CC}	Supply Current		20 45	mA	See Note 10 See Note 11
I_{CCSB}	Standby Supply Current		2	mA	See Note 12
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND
For Pins 25, 34, 37, 39 (WR CLOCK, DRUN, READ DATA, READ CLOCK)					
TRS	Rise Time		30	ns	0.9V to 4.2V

A.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = +5V ± 10%; GND = 0V

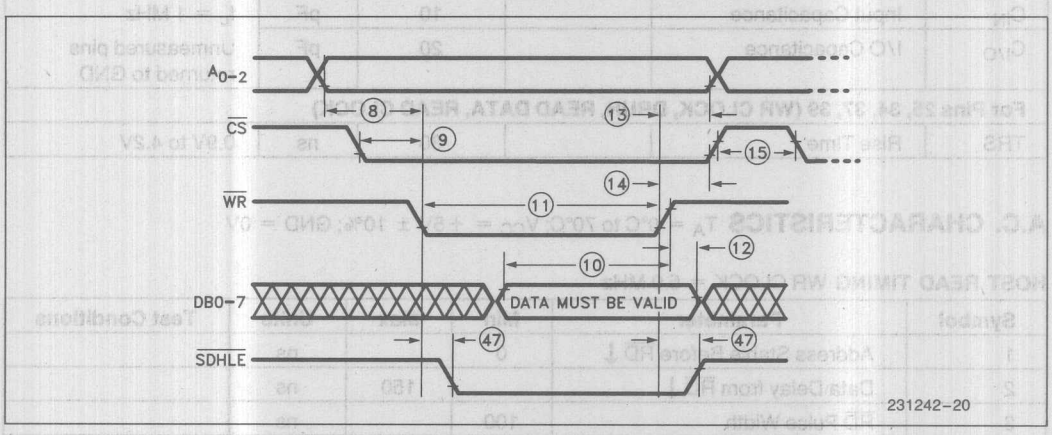
HOST READ TIMING WR CLOCK = 5.0 MHz

Symbol	Parameter	Min	Max	Units	Test Conditions
1	Address Stable Before $\overline{RD}$ ↓	0		ns	
2	Data Delay from $\overline{RD}$ ↓		150	ns	
3	$\overline{RD}$ Pulse Width	100		ns	
4	Data Valid after RD ↑	10	100	ns	
5	Address Hold Time after $\overline{RD}$ ↑	0		ns	
6	Read Recovery Time	300		ns	
7	$\overline{CS}$ Stable before $\overline{RD}$ ↓	0		ns	See Note 6



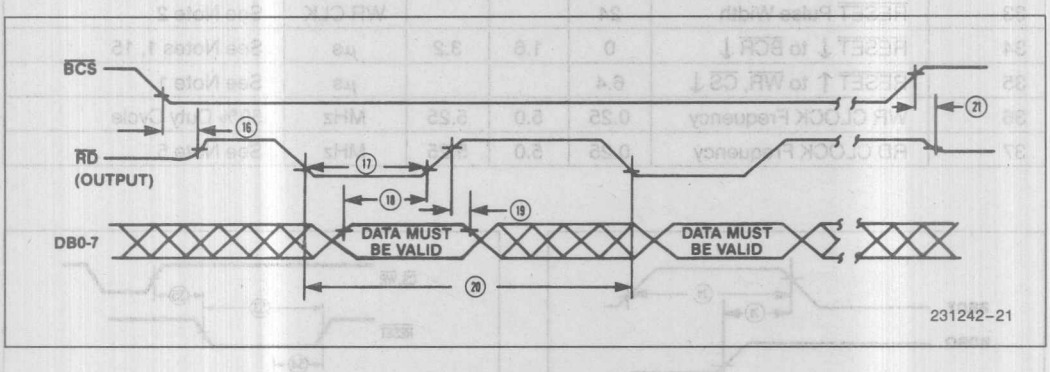
HOST WRITE TIMING WR CLOCK = 5.0 MHz

Symbol	Parameter	Min	Max	Units	Test Conditions
8	Address Stable Before $\overline{WR}\downarrow$	0		ns	
9	$\overline{CS}$ Stable Before $\overline{WR}\downarrow$	0		ns	
10	Data Setup Time Before $\overline{WR}\uparrow$	75		ns	
11	$\overline{WR}$ Pulse Width	100	10000	ns	
12	Data Hold Time After $\overline{WR}\uparrow$	0		ns	
13	Address Hold Time After $\overline{WR}\uparrow$	0		ns	
14	$\overline{CS}$ Hold Time After $\overline{WR}\uparrow$	0		ns	See Note 7
15	Write Recovery Time	300		ns	
47	$\overline{SDHLE}$ Propagation Delay	20	150	ns	



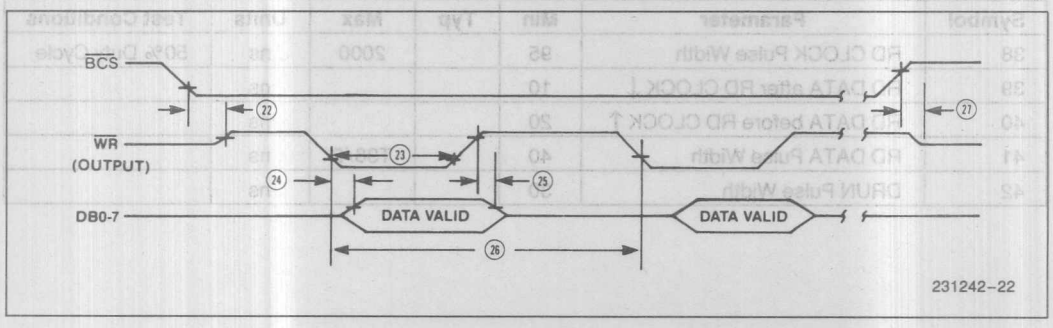
BUFFER READ TIMING (WRITE SECTOR COMMAND) WR CLOCK = 5.0 MHz

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
16	$\overline{BCS}$ ↓ to $\overline{RD}$ Valid	0		100	ns	
17	$\overline{RD}$ Output Pulse Width	300	400	500	ns	See Note 3
18	Data Setup to $\overline{RD}$ ↑	140			ns	
19	Data Hold from $\overline{RD}$ ↑	0			ns	
20	$\overline{RD}$ Repetition Rate	1.2	1.6	2.0	μs	See Note 8
21	$\overline{RD}$ Float from $\overline{BCS}$ ↑	0		100	ns	



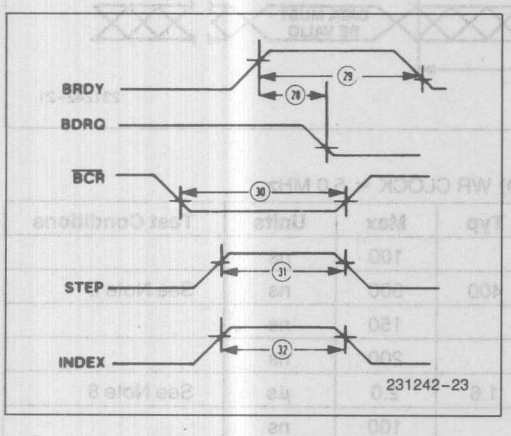
BUFFER WRITE TIMING (READ SECTOR COMMAND) WR CLOCK = 5.0 MHz

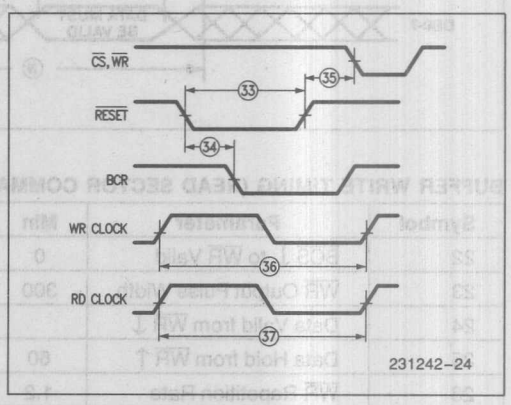
Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
22	$\overline{BCS}$ ↓ to $\overline{WR}$ Valid	0		100	ns	
23	$\overline{WR}$ Output Pulse Width	300	400	500	ns	See Note 3
24	Data Valid from $\overline{WR}$ ↓			150	ns	
25	Data Hold from $\overline{WR}$ ↑	60		200	ns	
26	$\overline{WR}$ Repetition Rate	1.2	1.6	2.0	μs	See Note 8
27	$\overline{WR}$ Float from $\overline{BCS}$ ↑	0		100	ns	



MISCELLANEOUS TIMING

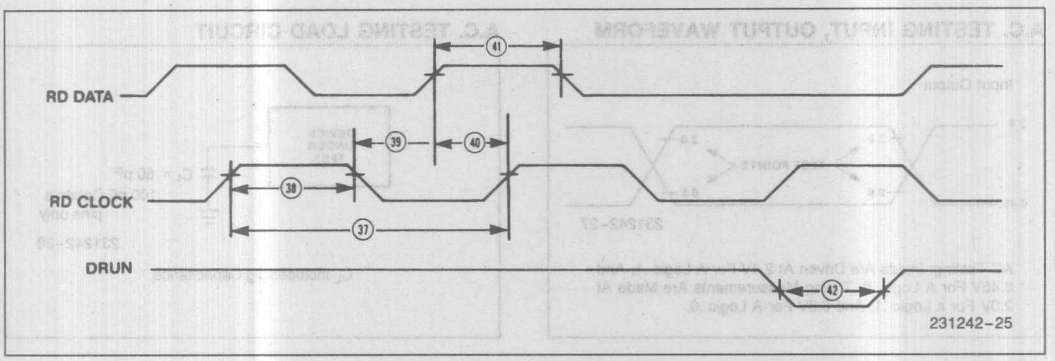
Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
28	BDRQ Reset from BRDY	20		200	ns	
29	BRDY Pulse Width	400			ns	See Note 4
30	$\overline{BCR}$ Pulse Width	1.4	1.6	1.8	μs	See Notes 9, 13, 15
31	STEP Pulse Width	1.5	1.6	1.7	μs	Step Rate = 3.2 μs/step
		7.6	8.0	8.4	μs	All other step rates, See Notes 14, 15
32	INDEX Pulse Width	500			ns	
33	$\overline{RESET}$ Pulse Width	24			WR CLK	See Note 2
34	$\overline{RESET}$ ↓ to $\overline{BCR}$ ↓	0	1.6	3.2	μs	See Notes 1, 15
35	$\overline{RESET}$ ↑ to $\overline{WR}$, $\overline{CS}$ ↓	6.4			μs	See Note 1
36	WR CLOCK Frequency	0.25	5.0	5.25	MHz	50% Duty Cycle
37	RD CLOCK Frequency	0.25	5.0	5.25	MHz	See Note 5





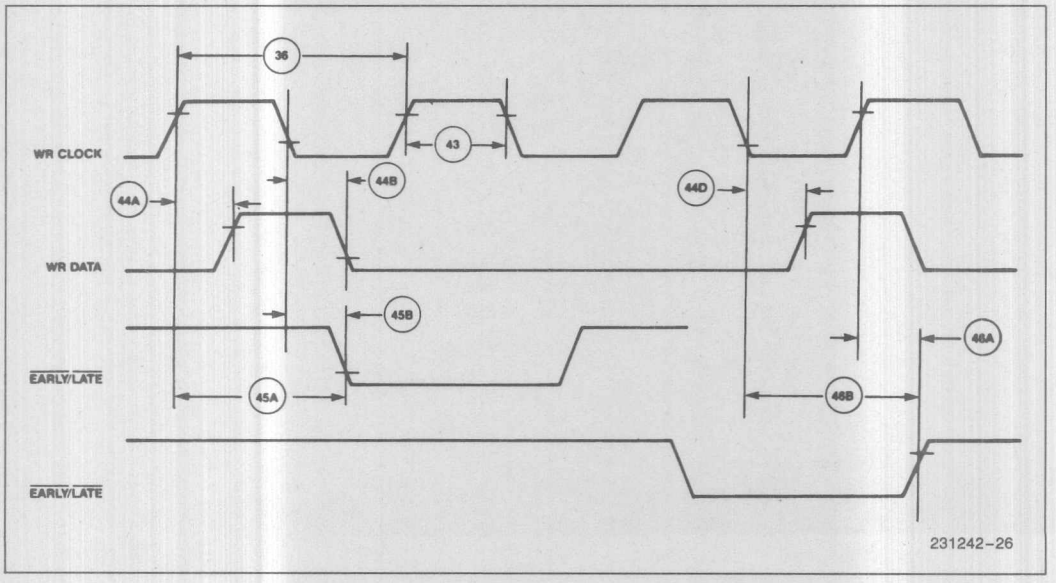
READ DATA TIMING WR CLOCK = 5.0 MHZ

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
38	RD CLOCK Pulse Width	95		2000	ns	50% Duty Cycle
39	RD DATA after RD CLOCK ↓	10			ns	
40	RD DATA before RD CLOCK ↑	20			ns	
41	RD DATA Pulse Width	40		T38/2	ns	
42	DRUN Pulse Width	30			ns	

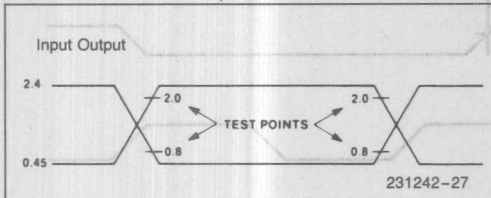


WRITE DATA TIMING WR CLOCK = 5.0 MHZ

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
43	WR CLOCK Pulse Width	95		2000	ns	50% Duty Cycle
	Propagation Delay					
44A	WR CLOCK ↑ to WR DATA ↑	10		65	ns	
44B	WR CLOCK ↓ to WR DATA ↓					
44D	WR CLOCK ↓ to WR DATA ↑					
45A	WR CLOCK ↑ to $\overline{\text{EARLY/LATE}}$ ↓	10		65	ns	
45B	WR CLOCK ↓ to $\overline{\text{EARLY/LATE}}$ ↓					
46A	WR CLOCK ↑ to $\overline{\text{EARLY/LATE}}$ ↑	10		65	ns	
46B	WR CLOCK ↓ to $\overline{\text{EARLY/LATE}}$ ↑					

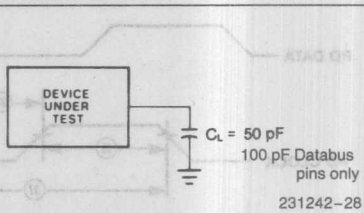


A.C. TESTING INPUT, OUTPUT WAVEFORM



AC Testing: Inputs Are Driven At 2.4V For A Logic .1, And 0.45V For A Logic .0. Timing Measurements Are Made At 2.0V For a Logic .1, And 0.8V For A Logic .0.

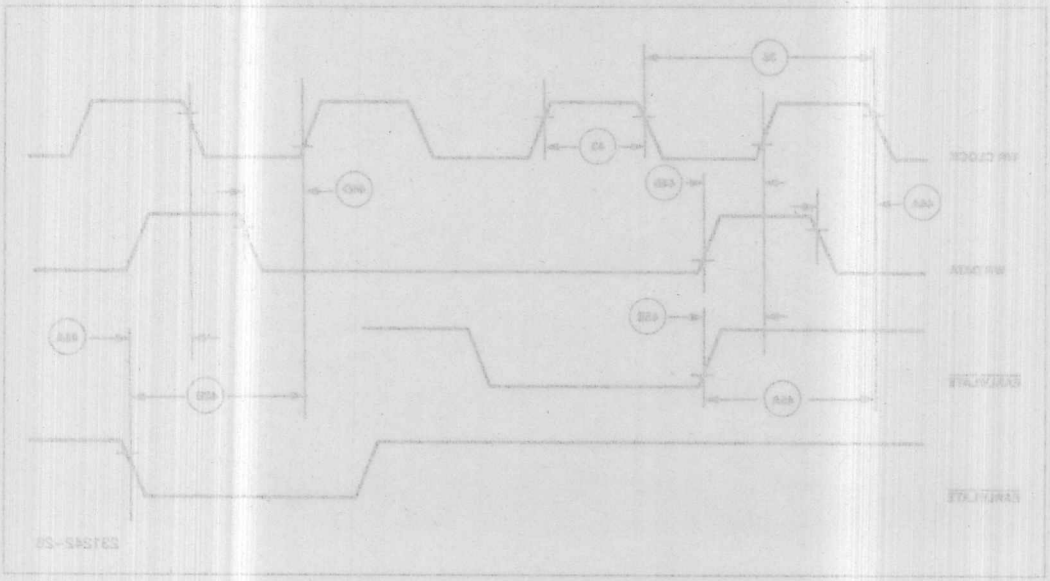
A.C. TESTING LOAD CIRCUIT



C_L Includes Jig Capacitance

NOTES

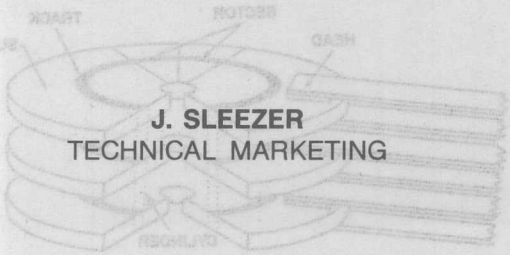
1. Based on WR CLOCK = 5.0 MHz
2. 24 WR CLOCK periods = 4.8 μs at 5.0 MHz.
3. 2 WR CLOCK periods ± 100 ns.
4. Previous restrictions on BRDY no longer apply. There are no restrictions on when BRDY may come. BRDY may be connected directly to BDRQ.
5. WR CLOCK Frequency = RD CLOCK Frequency ± 15%.
6. $\overline{RD}$ may be asserted before $\overline{CS}$ as long as it remains active for at least the minimum T3 pulse width after $\overline{CS}$ is asserted.
7. $\overline{WR}$ may be asserted before $\overline{CS}$ as long as it remains active for at least the minimum T11 pulse width after $\overline{CS}$ is asserted.
8. 8 WR CLOCK periods ±2 WR CLOCK periods.
9. 8 WR CLOCK periods ±1 WR CLOCK period.
10. V_{IL} = GND, V_{IH} = V_{CC}, Outputs Open.
11. V_{IL} = 0.8V, V_{IH} = 2.0V, Outputs Open.
12. WR CLOCK & RD CLOCK = DC, V_{IL} = 0V, V_{IH} = V_{CC}, all output open, $\overline{CS}$ inactive.
13. This specification is for $\overline{BCR}$ pulse width during command execution. $\overline{BCR}$ is also triggered by $\overline{RESET}$. In this case, $\overline{BCR}$ pulse width is greater than $\overline{RESET}$ pulse width.
14. 40 WR Clocks ±2.
15. Specification represents actual functionality of 82064 and WD2010. Previous datasheets contain typographical errors.



NOTE

November 1986

Multimodule™ Winchester Controller Using the CHMOS 82064



J. SLEEZER
TECHNICAL MARKETING

Order Number: 231927-001

1.0 INTRODUCTION

The 82064 Winchester Disk Controller (WDC) was developed to ease the complex task of interfacing Winchester disk drives to microprocessor systems. Specifically, the 82064 WDC interfaces to drives that conform to the ST506 specification, which is the dominant interface for 5¼ inch drives. This Application Note provides some background on the 82064 WDC, the drive interfaces and general software routines. It concludes with a design example using the 82064 WDC interfaced to the SBX™ bus. Appendix B contains the listing of the software necessary to operate this controller board.

1.1 ST506 Winchester Drive Overview

Since the 82064 WDC interfaces only to drives conforming to the ST506 specification, this overview will limit itself to those drives. A summary of the ST506 specification is shown in Appendix A for those who are not familiar with it. The ST506 Winchester Disk contains from 1 to 8 hard disks (or platters) with the average being 2 to 3 disks. These disks are made from aluminum (hence the term hard disk) and are coated with some type of recording media. The recording media is typically made of magnetic-oxide, which is similar to the material used on floppy disks and cassette tapes. Each side of a hard disk is coated with recording media and each side can store data. Each surface of a disk has its own read/write head.

Hard disk drives are sealed units because the R/W heads actually fly above the disk surface at about 8 to 20 microinches. A piece of dust or dirt, which appears as a boulder to the gap between the heads and the disk surface, will wreak havoc upon the disk media.

The R/W heads are mechanically connected together and move as a single unit across the surface of the disk. There are 2 basic methods for positioning the heads. The first is with stepper motors, which is the most common method and is also used on most floppy disk drives. These positioners are used mainly because of their low cost.

The second method of positioning the heads is to use a voice-coil mechanism. These units do not move in steps but swing across the disk. These mechanisms generally permit greater track density than steppers, but also require complex feedback electronics which increases the cost of the drive. Generally, voice-coil head positioners use closed loop servo positioning, as compared to the open loop positioning used with stepper motors.

The surface of a disk is divided logically into concentric circles radiating from the center as shown in Figure 1. Each concentric circle is called a track.

The group of tracks, all in the same position, on all of the disks (platters) in the drive is collectively called a cylinder. The number of tracks on a surface (which affects storage density) is determined by the head positioners. Typically, stepper head positioners have fewer tracks than drives that use a voice coil positioner. Which type of positioner is used is irrelevant to the 82064 as positioners are part of the drive electronics. The 82064 can access up to 2048 tracks per surface.

Once the surface is divided into cylinders it is further divided radially (as with a pie). The area between the radial spokes is referred to as a sector. The number of sectors per track is determined by many variables, but is basically determined by the number of data bytes and the length of the ID field (which locates a sector). Figure 2 shows one manufacturer's specifications for their drive. The manufacturer formats the drive with 32–256 byte sectors per track. Alternatively, the drive could be reformatted to contain 17–512 byte sectors per track. This second option has fewer sectors per track but stores more data. Determining how many bytes each sector contains is done by extensive analysis of the hardware and operating system. The 82064 WDC is programmable for sector size during formatting.

The order in which sectors are logically numbered on the track is called interleaving. An interleave factor of four would have three sectors separating logically sequential sectors. Starting at the index pulse, an example of four way interleaving is:

Sector 1, Sector X, Sector Y, Sector Z, Sector 2, Sector . . .

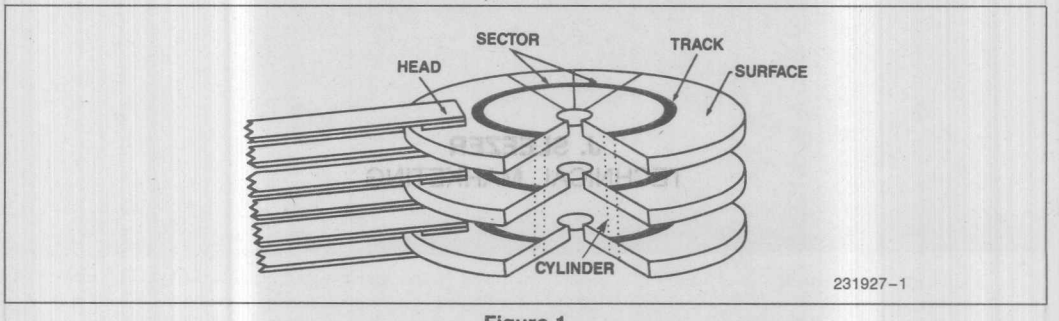


Figure 1

Capacity	
Unformatted	
Per Drive	6.38 Megabytes
Per Surface	1.59 Megabytes
Per Track	10416 Bytes
Formatted	
Per Drive	5.0 Megabytes
Per Surface	1.25 Megabytes
Per Track	8192 Bytes
Per Sector	256 Bytes
Sectors per Track	32
Transfer Rate	5.0 Megabits per second
Access Time	
Track to Track	3 ms
Average (Inc. Settle)	170 ms
Maximum (Inc. Settle)	500 ms
Settling Time	15 ms
Average Latency	8.33 ms
Functional Specifications	
Rotational speed	3600 rpm ± 1%
Recording density	7690 bpi max
Flux density	7690 fci
Track density	255 tpi
Cylinders	153
Tracks	612
R/W Heads	4
Disks	2

Figure 2. A Typical Drive Specification

Interleaving is used primarily because one sector at a time is transferred from disk to sector buffer to system RAM. This transferring of data takes time, and causes a delay between the first sector transferred and sectors that follow it. Without interleaving, the delay in transferring data would result in sectors on the disk rotating past the heads before they could be read. The operating system would then have to wait one disk revolution to get to the next sector (a 16.7 msec delay). With interleaved sectors, the next logical sector would be positioned beneath the heads after the previous sector of data had been transferred to the system RAM. Interleaving unfortunately slows down the overall transfer rate from the disk. A 5 Mbit/second transfer rate averages out to a 1.25 Mbit/second transfer rate when many sectors are transferred with four way interleaving. Again, how much interleaving to use is determined by extensive hardware/software benchmarking.

Whenever data is stored on a multiple platter disk drive, the same track on all surfaces whould be used before repositioning the heads to another track. Repositioning the heads generates a longer delay due to the mechanical delay of moving the heads. Switching to another head incurs no mechanical positioning delay. Only one head can be selected at a time.

Hard disk drives tend to be faster than floppies for two reasons. The speed at which the disk spins is about 10 times faster than the floppy (a floppy spins at 360 rpm for the popular double density disk drives). This yields an immediate one-tenth reduction in access times for the same size drive. While both ST506 drives and floppies use stepper motors, the steppers utilized by the hard disk drives are approximately twice as fast as those used by floppies.

2.0 82064 WINCHESTER DISK CONTROLLER

The 82064 WDC provides most of the functions necessary to interface between a microprocessor and an ST506 compatible disk drive. The 82064 converts the high level commands and parallel data of a microprocessor bus into ST506 compatible disk control signals and serial MFM encoded data. This section presents a detailed description of the 82064 and a discussion of various techniques which can be used to interface the 82064 to a microprocessor.

The internal structure of the 82064 is divided into several sections as shown in Figure 3. They are:

1. the microprocessor interface which includes the status and task registers;
2. sector buffer control;
3. the drive interface;
4. the data transfer section, which includes the MFM encoding/decoding of microprocessor data;
5. and CRC/ECC generation and checker.

2.1 Clock Inputs

The 82064 has two clock inputs: read clock (RD CLOCK) and write clock (WR CLOCK). The PLA controller, the processor interface, buffer control and MFM encoding sections operate off the WR CLOCK input. The RD CLOCK input is used only for decoding the MFM data stream. The clocks may be asynchronous to one another. Both clocks have non-TTL compatible inputs. The easiest method to interface to TTL requires a pull-up resistor to satisfy their input voltage needs. The resistor's value must be compatible with the VIL specification of these pins. See the Pin Descriptions Section for more specific information.

2.2 Microprocessor Interface

The microprocessor interface of the 82064 contains the control logic which permits commands and data to be transferred between the host and the 82064. The interface consists of an 8 bit, tri-state, bidirectional data bus; the task registers; a 3 to 8 address decoder for selecting one of the seven registers; and the general read, write, and chip select logic. Externally, the 82064 expects a buffer equal in size to a sector on the disk, and tri-state

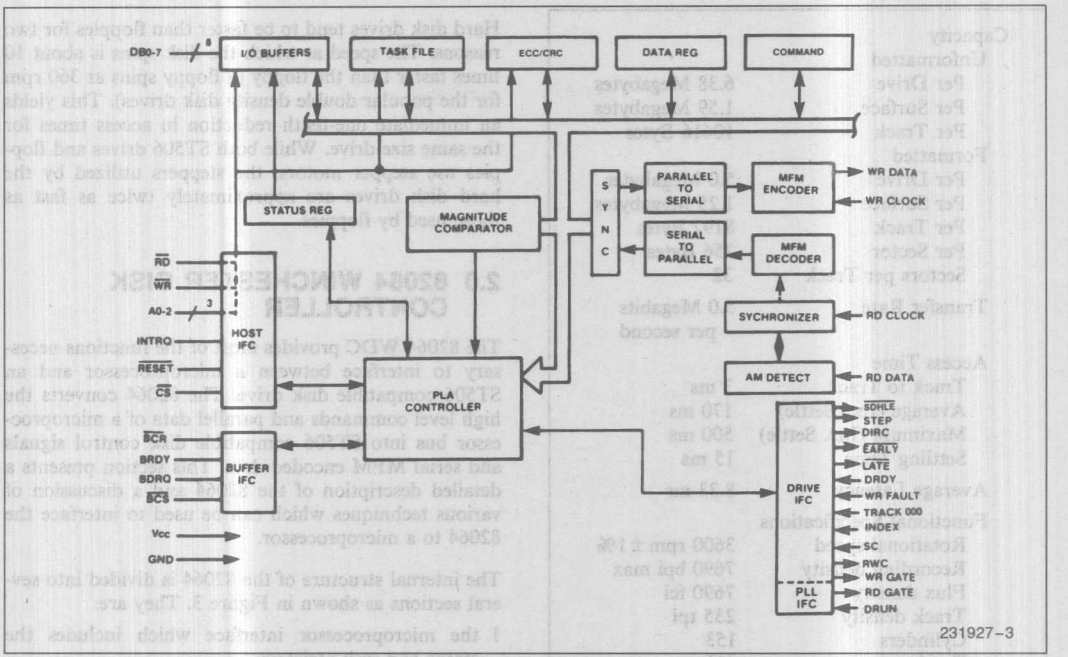


Figure 3. 82064 Internal Block Diagram

transceivers between the sector buffer and the microprocessors data bus in order to isolate itself from the microprocessor during disk data transfers.

A0–A2, Data Bus

These three address lines are active high signals and select one of the seven register locations in the 82064. They are not latched internally. If the three addresses are equal to 0 and the 82064 is selected, the data bus is kept tri-stated to ease interfacing to a sector buffer. The 82064's data bus is controlled by both the microprocessor and the 82064. The microprocessor has control for loading the registers and command. During disk reads or writes, control switches to the 82064 so that it may access the local sector buffer when transferring data between the disk and the buffer.

$\overline{RD}$, $\overline{WR}$, $\overline{CS}$

The chip select ($\overline{CS}$) is typically decoded from the higher order address lines. $\overline{CS}$ only permits data to be placed into, or read from, the 82064's task registers. Once a disk operation starts, $\overline{CS}$ no longer efffects the 82064. $\overline{RD}$ and $\overline{WR}$ are bidirectional lines and are used to read or write the 82064's registers by the host microprocessor and are valid only if $\overline{CS}$ is present. The 82064 will drive $\overline{RD}$ and $\overline{WR}$ when transferring data between the sector buffer and the disk. A signal is provided to tri-state the $\overline{RD}$ and $\overline{WR}$ lines from the host during a buffer access. This is covered in the Sector Buffer Control Section.

Interrupts

An interrupt is issued at the end of all commands, and the interrupt is cleared by reading any register. For the Read Sector command only, the 82064 allows the user the option of an interrupt either at the termination of the command, as is the case with all other commands, or when data needs to be transferred to the host from the sector buffer. This is discussed further in the Interrupt Mode Section. When selecting the data transfer option, the interrupt line will go active at the same time as the BDRQ line and the interrupt will be removed only when the proper handshake occurs with the sector buffer.

Task Registers

The Task Register File contains the command, status, track number, sector number, and other information necessary to properly execute a command. These registers are accessed with A0–A2, $\overline{RD}$ (or $\overline{WR}$), and $\overline{CS}$ being valid and are not cleared by a reset. The registers are covered in detail in the Task Register File Section.

2.3 Sector Buffer Control

The 82064 was designed to operate with an external buffer equal in size to one sector. To ease the design-in of this buffer, the 82064 provides all of the control signals it needs to operate the buffer. This buffer must be isolated from the system bus, using tri-state buffers, during disk transfers to prevent contention during the period that the 82064 is accessing the buffer. A sector buffer is generally used to ease interfacing to the system due to the high disk data rates (625 kbytes/sec), although it is not required.

$\overline{BCS}$

The Buffer Chip Select ($\overline{BCS}$) line goes active whenever the 82064 is accessing the sector buffer. This signal should remove the microprocessors ability to access the 82064 and sector buffer and must enable the sector buffer for use by the 82064.

At a 5 Mbit/sec disk data rate, the 82064 will access the buffer every 1.6 microseconds (8 bits × 200 ns/bit). $\overline{BCS}$ will remain low the entire time the 82064 is accessing the buffer. The 82064 will pulse the appropriate $\overline{RD}$ or $\overline{WR}$ line for each byte transferred.

$\overline{BCR}$

Buffer Counter Reset ($\overline{BCR}$) goes active each time that $\overline{BCS}$ changes state. Its purpose is to reset the address counter of the sector buffer back to zero before and after the 82064 uses the sector buffer. Its function is optimized for single sector transfers. Multiple sector transfers should use a software controlled buffer counter reset and not use $\overline{BCR}$ as the sector buffer will be reset to the beginning after each sector is transferred.

BDRQ, BRDY

Buffer Data Request (BDRQ) and Buffer Ready (BRDY) provide the handshake needed to transfer data between the sector buffer and the host. BDRQ signals that data must be moved to/from the sector buffer and the host. BRDY has two functions. Once the transfer signaled by BDRQ is finished, asserting BRDY will inform the 82064 that the transfer is completed and that it may finish executing the command. BRDY is also used in multiple sector commands. BRDY going high during a multiple sector transfer indicates that the buffer is full (or empty—depending upon the command) and the transfer should wait until the buffer is serviced. The sector that was being transferred will finish and the 82064 will deactivate $\overline{BCS}$ and activate BDRQ. The host microprocessor must then transfer the data between the buffer and system memory. When this transfer is finished, asserting BRDY will cause the 82064 to resume the command.

The handshaking between BDRQ and BRDY occurs only in full sector increments—not on a byte basis. A high on BDRQ indicates a full sector's worth of data is required; BRDY going high indicates a full sector of data is available to the 82064 without interruption.

Only the rising edge of BRDY is valid. A falling edge may occur at any time without effect. $\overline{BCR}$ will pulse and $\overline{BCS}$ will go active eight byte times (8 bytes × 8 bits/byte × 200 ns/bit = 12.8 microseconds) before the first data byte is transferred from the sector buffer to the disk.

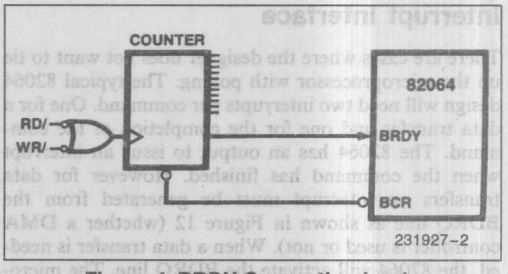


Figure 4. BRDY Generation Logic

2.4 Data Transfer Logic

This section of the 82064 is responsible for conversion of serial disk data to parallel data (and vice versa); encoding/decoding of the disk's MFM serial bit stream; and detecting the address mark.

Polled Interface

Since the 82064 isolates itself from the host during several commands, the host cannot read the status register during some periods to determine what course should be taken. In Figure 10, trying to read the status register when $\overline{BCS}$ is active will return indeterminate data. To prevent the microprocessor from reading this indeterminate data, a hardware generated "Busy" pattern should be driven onto the data bus if $\overline{BCS}$ is active. This is shown in Figure 11. The status register contains a data request (DRQ) bit whose timing is equal to the BDRQ output signal, thus making a polled operation possible. DRQ will stay set in the status register until a BRDY is generated.

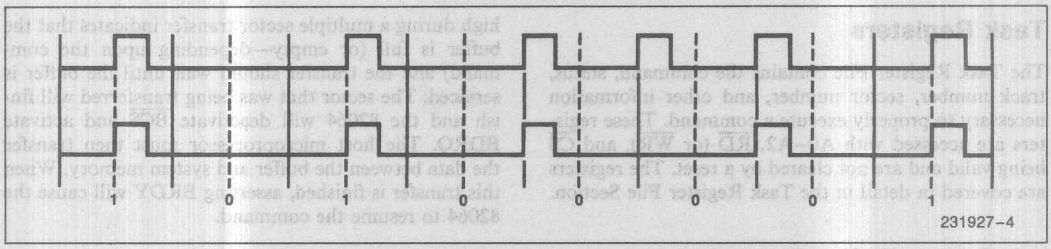


Figure 5. Data Address Mark

One design issue with the polled interface occurs when the microprocessor is polling the status and the 82064 deactivates $\overline{BCS}$. The microprocessor would normally read the hardware busy pattern. If $\overline{BCS}$ is deasserted, the hardware pattern is disabled and the microprocessor will start to read the real status register. The read cycle may almost be finished, and the read access period of the 82064 will not be satisfied. The data returned to the microprocessor will be invalid.

Interrupt Interface

There are cases where the designer does not want to tie up the microprocessor with polling. The typical 82064 design will need two interrupts per command. One for a data transfer and one for the completion of the command. The 82064 has an output to issue an interrupt when the command has finished. However for data transfers an interrupt must be generated from the BDRQ line as shown in Figure 12 (whether a DMA controller is used or not). When a data transfer is needed, the 82064 will activate the BDRQ line. The microprocessor will be interrupted and do the data transfer function. BDRQ will stay active until BRDY is generated, so the system must either use edge triggered interrupts or must not write the end-of-interrupt byte until BDRQ is removed (this is true of Intel's 8259A).

MFM Encoding/Decoding

The MFM encoding section will receive 8 bit parallel data when a valid command has been recognized and BRDY has gone high. The parallel data is first serialized and converted to an intermediate, NRZ encoded, data stream. The serial NRZ data is sent to the MFM encoding section and then transferred to the disk. Decoding of the MFM bit stream (during disk reads) happens in reverse order.

The control logic operates off the write clock (WR CLOCK) running at a frequency of the desired tranfer rate. The MFM decoding portion operates off of the read clock (RD CLOCK) input, which is supplied by an external phase lock loop. The two clocks need not be synchronized to each other. Data is written (and hence read) with the most significant bit first.

Address Mark Detector

The address mark is a unique 2 byte code written at the beginning of each ID field and data field. This address mark serves two purposes. It tells the controller what type of data is about to be received so that internal computations can be performed, and to ensure that ID fields are not sent to the host. The second purpose is to align the serial data back to the original 8 bit boundaries that existed when data was written (there are no byte boundaries on a disk).

An address mark is always preceded by the all zeros synchronization field. The 82064 starts comparing the incoming data stream when the synchronization field ends. A high speed comparator is used since the 82064 does not yet know where the proper byte boundaries are. When a proper comparison of the address mark is made the controller starts assembling bytes, starting with the second byte of the address mark.

The first byte of the address mark is an "A1" Hex, but purposely violates the MFM encoding rules by removing a clock pulse. In Figure 5, the first example is of a normal MFM encoded A1H. The second example is of the address mark and shows the missing clock pulse. The non-MFM compatible A1 is to prevent the host from issuing a similar data byte and possibly confusing detection logic.

The second byte specifies either an ID or data field and is encoded according to normal MFM rules. It is either an "F8" Hex for a data field, or "FC" through "FF" for an ID field. The different values correspond to a range of cylinders on the drive in increments of 256 tracks. The 82064 makes no use of this information, but writes it for compatibility with the ST506 specification during formatting.

PLA Control

The PLA Controller interprets command sent by the microprocessor. Its operation is synchronized to the WR CLOCK input. The PLA controller is started when a command is written into the command register. It generates control signals and operates in a handshake mode when communicating with the MFM decoding block.

Magnitude Comparator

A 10 bit magnitude comparator is used to calculate the direction and number of step pulses needed to move the head from the present cylinder position to the desired position. A separate high speed equivalence comparator is used to compare ID field bytes when searching for a sector ID field.

2.5 CRC/ECC Generator and Checker

The 82064 provides two options for protecting the integrity of the data field. The data field may have either a CRC (SDH register, bit 7 = 0), or a 32-bit ECC (SDH register, bit 7 = 1) appended to it. The ID field is always protected by a CRC.

CRC Generation/Checking

The CRC generator computes and checks the cyclic redundancy check bytes that are appended to the ID and data fields. CRC generation/checking is always done on ID fields. Data fields have a choice between 82064 CRC, internal ECC or externally supplied ECC. The CRC mode is chosen by setting bit 7 of the SDH register low. The CRC mode provides a means of verifying the accuracy of the data read from the disk, but does not attempt to correct it. The CRC generator computes and checks cyclic redundancy check characters that are written and read from the disk after the ID and data fields.

The generator polynomial for the CRC-CCITT (CRC-16) code is:

$$x16 + x12 + x5 + 1 = (x + 1)(x15 + x14 + x13 + x12 + x4 + x3 + x2 + x + 1)$$

The code's capability is as follows:

a) Detects all occurrences of an odd number of bits in error.

b) Detects all single, double, and triple bit errors if the record length (including check bits) is less than 32,767 bits.

c) Detects all single-burst errors of sixteen bits or less.

d) Detects 99.99695% of all possible 17 bit burst errors, and 99.99847% of all possible longer burst, assuming all errors are possible and equally probable.

The CRC code has some double-burst capability when used with short records (sectors). For a 256 byte sector the code will detect double-bursts as long as the total number of bits in error does not exceed 7.

If the CRC character generated while reading the data does not equal the one previously written, an error exists. If an ID field CRC error occurs the "ID not found" bit in the error register will be set. If a data field CRC error occurs the "ECC/CRC" bit in the error register will be set.

ECC Generation/Checking

The ECC mode is only applicable to the data field. It provides the user with the ability to detect and correct errors in the data field automatically. The commands and registers which must be considered when ECC is used are:

1. SDH Register, bit 7 (CRC/ECC)
2. READ SECTOR Command, bit 0 (T)
3. READ SECTOR and WRITE SECTOR Commands, bit 1 (L)
4. COMPUTE CORRECTION Command
5. SET PARAMETER Command
6. STATUS Register, bit 2—error correction successful
7. STATUS Register, bit 0—error occurred
8. ERROR Register, bit 6—uncorrectable error

To enable the ECC mode, bit 7 of the SDH register must be set to one.

Bit 0 (T) of the READ Command controls whether or not error correction is attempted. When T = 0 and an error is detected, the 82064 tries up to 10 times to correct the error. If the error is successfully corrected, bit 2 of the STATUS Register is set. The host can interrogate the status register and detect that an error occurred and was corrected. If the error was not correctable, bit 6 of the ERROR Register is set. If the correction span was set to 5 bits, the host may now execute the SET PARAMETER Command to change the correction span to 11 bits, and attempt the read again. If the error persists, the host can read the data, but it will contain errors.

When T = 1 and an error is detected, no attempt is made to correct it. Bit 0 of the STATUS Register and bit 6 of the ERROR Register are set. The user now has two choices:

1. Ignore the error and make no attempt to correct it.
2. Use the COMPUTER CORRECTION Command to determine the location and pattern of the error, and correct it within the user's program.

When the COMPUTE CORRECTION Command is implemented, it must be done before executing any command which can alter the contents of the ECC Register. The READ SECTOR, WRITE SECTOR,

SCAN ID, and FORMAT Commands will alter this register and correction will be impossible. The COMPUTE CORRECTION Command may determine that the error is uncorrectable, at which point the error bits in the STATUS and ERROR Registers are set.

Although ECC generation starts with the first bit of the F8H byte in the data ID field, the actual ECC bytes written will be the same as if the A1H byte was included. The ECC polynomial used is:

$$X^{32}+X^{28}+X^{26}+X^{19}+X^{17}+X^{10}+X^{6}+X^{2}+1$$

For automatic error correction, the external sector buffer must be implemented with a static RAM and counter, not with a FIFO.

The SET PARAMETER Command is used to select a 5-bit or 11-bit correction span.

2.6 Drive Interface

The drive interface of the 82064 contains the logic that makes possible the storage and reliable recovery of data. This interface consists of the drive and head select logic, the disk control signals, and read and write data logic as shown in Figure 6. This section describes the external circuitry which is required to complete the 82064's drive interface.

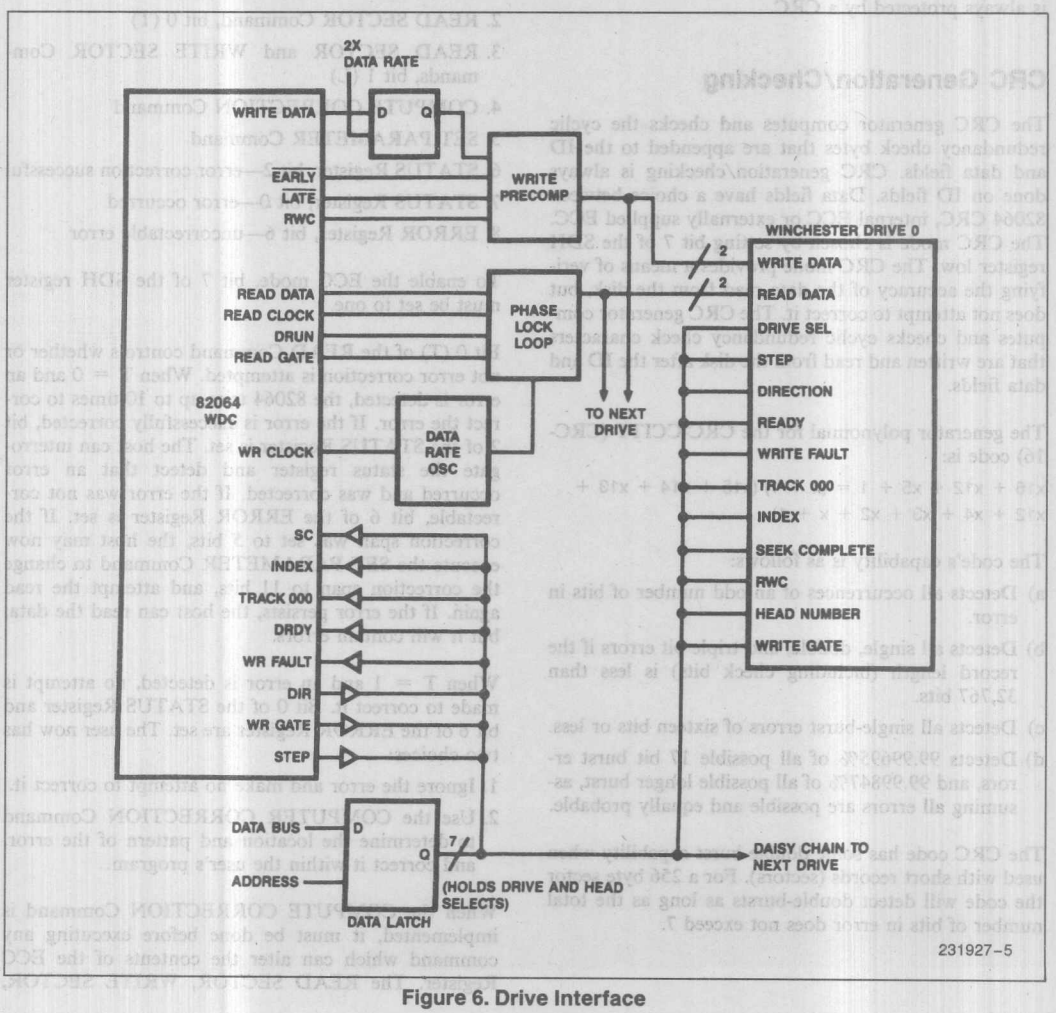


Figure 6. Drive Interface

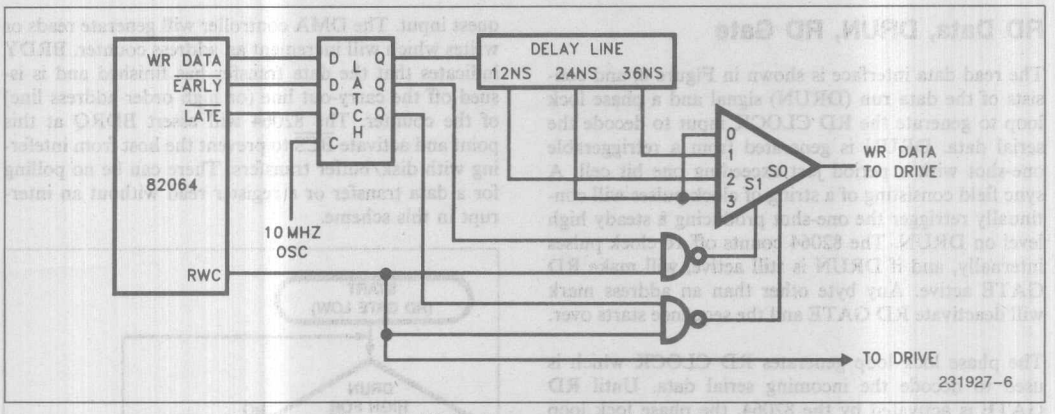


Figure 7. Write Precompensation Logic

Drive/Head Select

The 82064 has no outputs for selecting the head or drive. Therefore these signals must be generated by the user as shown in Figure 6. Data bits 0–4 should be latched whenever the SDH register is written. Bits 0–2 would then be driven onto the drive cable with open collector buffers. Bits 3 and 4 would be decoded after being latched, then buffered for the cable. The head information written to the 82064's SDH register is used to write the proper ID fields during formatting. Changing the drive bits in the SDH register will cause a Scan ID to be performed by the 82064 to update non user accessible registers.

Drive Control

The drive control (STEP, DIR, WR FAULT, TRACK 000, INDEX, SC, RWC, and WR GATE) signals are merely conditioned for transmission over the drive cable. The purpose of each pin can be found in the section on Pin Descriptions and their use in the Command Section.

WR DATA, $\overline{\text{EARLY}}$, $\overline{\text{LATE}}$

Figure 7 is a diagram of the interface required on the write data line. The final stage of the MFM encoding requires applying the WR DATA to an external flip-flop clocked at 10 MHz. The 82064 monitors the serial write data output for particular bit patterns that require precompensation to prevent bit shifting. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are active on all cylinders and will normally require that RWC be factored into them to activate the data precompensation on the proper cylinder.

A delay line is required to generate the delayed data for precompensation since the actual delay varies between drive manufacturers. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ go active in the same clock period that generates the data bit to be shifted.

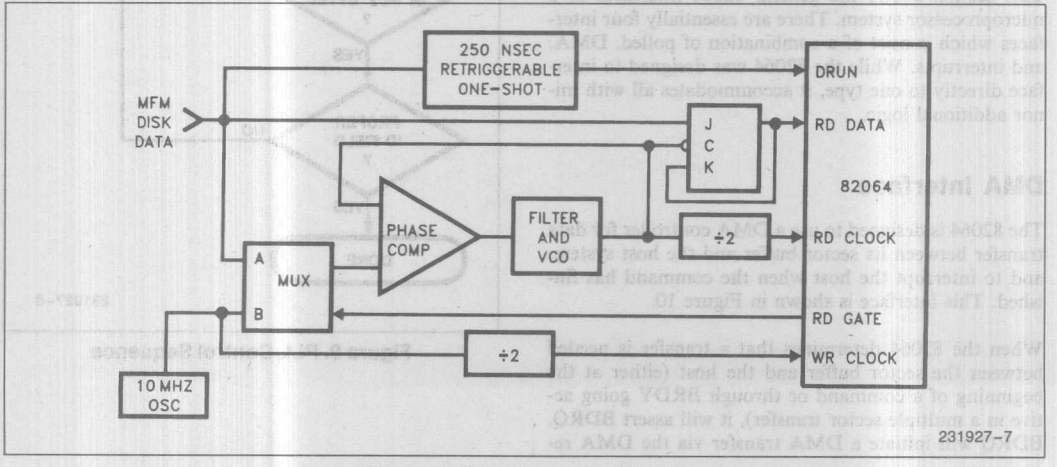


Figure 8. Data Separator Circuit

RD Data, DRUN, RD Gate

The read data interface is shown in Figure 8, and consists of the data run (DRUN) signal and a phase lock loop to generate the RD CLOCK input to decode the serial data. DRUN is generated from a retriggerable one-shot with a period just exceeding one bit cell. A sync field consisting of a string of clock pulses will continually retrigger the one-shot producing a steady high level on DRUN. The 82064 counts off 16 clock pulses internally, and if DRUN is still active, will make RD GATE active. Any byte other than an address mark will deactivate RD GATE and the sequence starts over.

The phase lock loop generates RD CLOCK which is used to decode the incoming serial data. Until RD GATE is activated by the 82064, the phase lock loop (PLL) should be locked onto a local 10 MHz clock to minimize PLL lock-up times. When RD GATE is activated, the PLL starts locking onto the incoming data stream, which should consist of the all zeros sync field. Once the PLL locks onto this synch field, the 82064 will start examining the serial data for a non-zero byte. A non-zero byte will be indicated by DRUN dropping since the address mark follows the sync field and is an "A1" Hex. This sequence is shown in Figure 9. If the address mark is detected, and if it was preceded by at least 9 bytes of zeros, RD GATE will stay active. The 82064 will then assemble bytes of data, and ensure the proper ID field is found. If a non-zero or non-address mark byte was detected, RD GATE will go inactive for a minimum of 2 byte times. If a data field or the wrong ID field is detected, or the ID field was not preceded by 8 bytes of zeros, then RD GATE goes inactive and the sequence starts over with the 82064 examining the DRUN input.

2.7 Microprocessor Interfaces

This section shows the general 82064 interfaces to a microprocessor system. There are essentially four interfaces which consist of a combination of polled, DMA, and interrupts. While the 82064 was designed to interface directly to one type, it accommodates all with minor additional logic.

DMA Interface

The 82064 is designed to use a DMA controller for data transfer between its sector buffer and the host system, and to interrupt the host when the command has finished. This interface is shown in Figure 10.

When the 82064 determines that a transfer is needed between the sector buffer and the host (either at the beginning of a command or through BRDY going active in a multiple sector transfer), it will assert BDRQ. BDRQ will initiate a DMA transfer via the DMA request input. The DMA controller will generate reads or writes which will increment an address counter. BRDY indicates that the data transfer has finished and is issued off the carry-out line (or high order address line) of the counter. The 82064 will assert BDRQ at this point and activate $\overline{BCS}$ to prevent the host from intefering with disk/buffer transfers. There can be no polling for a data transfer or a register read without an interrupt in this scheme.

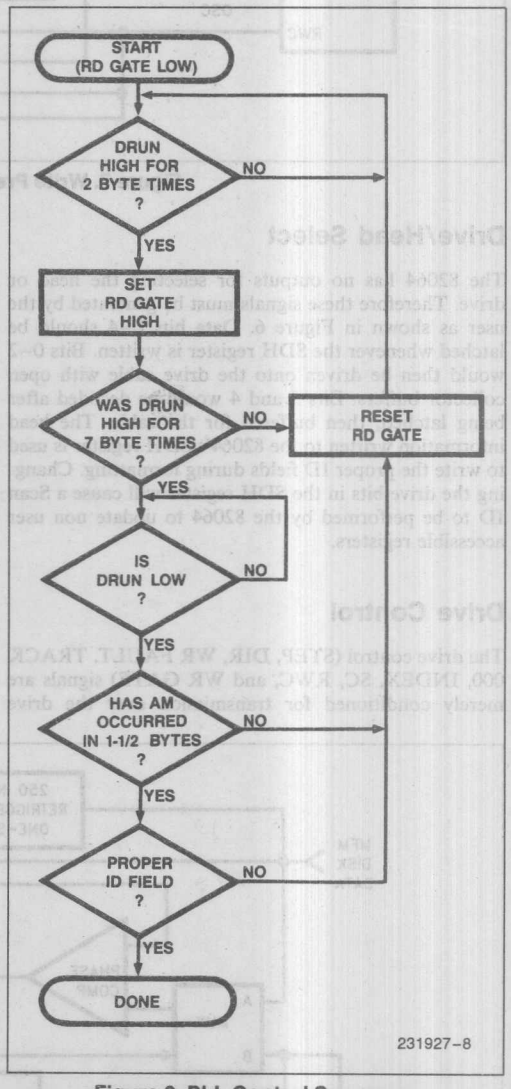


Figure 9. PLL Control Sequence

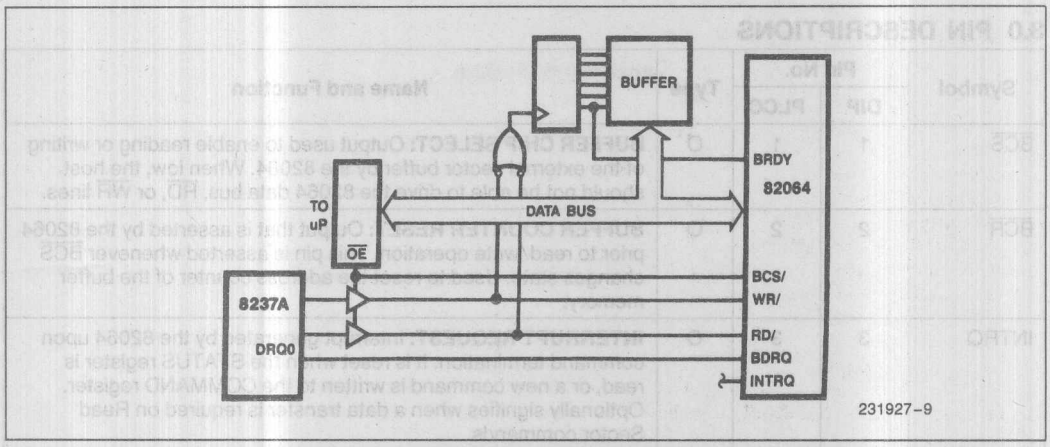


Figure 10. 82064 DMA Interface

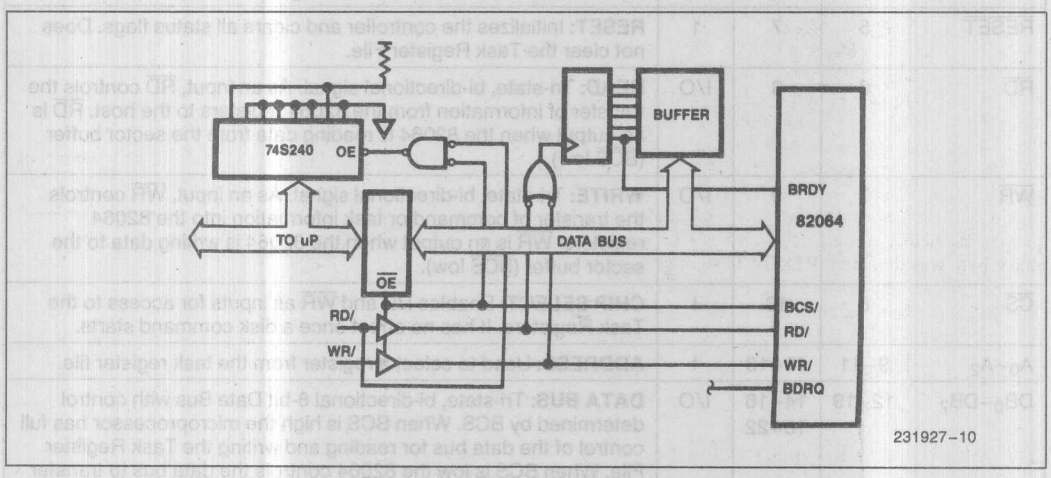


Figure 11. 82064 Polled Interface

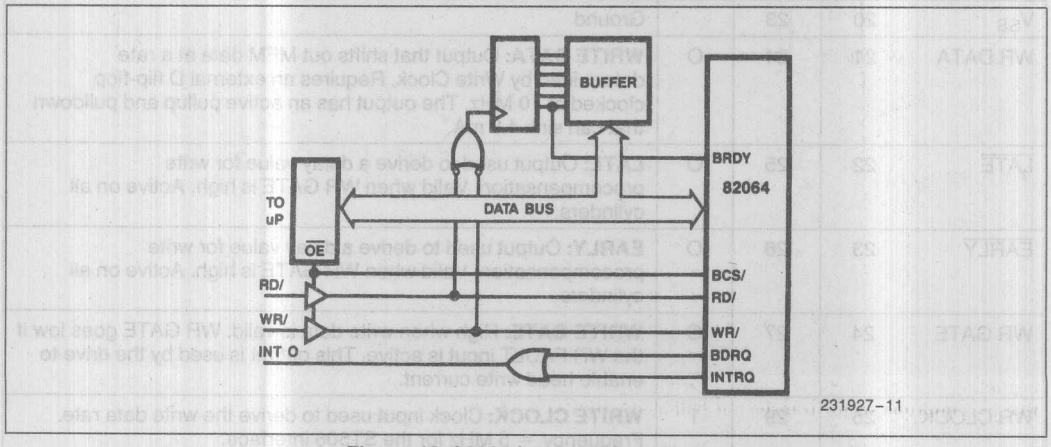


Figure 12. 82064 Interrupt Interface

3.0 PIN DESCRIPTIONS

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
$\overline{BCS}$	1	1	O	**BUFFER CHIP SELECT:** Output used to enable reading or writing of the external sector buffer by the 82064. When low, the host should not be able to drive the 82064 data bus, $\overline{RD}$, or $\overline{WR}$ lines.
$\overline{BCR}$	2	2	O	**BUFFER COUNTER RESET:** Output that is asserted by the 82064 prior to read/write operation. This pin is asserted whenever $\overline{BCS}$ changes state. Used to reset the address counter of the buffer memory.
INTRQ	3	3	O	**INTERRUPT REQUEST:** Interrupt generated by the 82064 upon command termination. It is reset when the STATUS register is read, or a new command is written to the COMMAND register. Optionally signifies when a data transfer is required on Read Sector commands.
$\overline{SDHLE}$	4	4	O	$\overline{SDHLE}$ is asserted when the SDH register is written by the host.
$\overline{RESET}$	5	7	I	**RESET:** Initializes the controller and clears all status flags. Does not clear the Task Register File.
$\overline{RD}$	6	8	I/O	**READ:** Tri-state, bi-directional signal. As an input, $\overline{RD}$ controls the transfer of information from the 82064 registers to the host. $\overline{RD}$ is an output when the 82064 is reading data from the sector buffer ($\overline{BCS}$ low).
$\overline{WR}$	7	9	I/O	**WRITE:** Tri-state, bi-directional signal. As an input, $\overline{WR}$ controls the transfer of command or task information into the 82064 registers. $\overline{WR}$ is an output when the 82064 is writing data to the sector buffer ($\overline{BCS}$ low).
$\overline{CS}$	8	10	I	**CHIP SELECT:** Enables $\overline{RD}$ and $\overline{WR}$ as inputs for access to the Task Registers. It has no effect once a disk command starts.
A_0–A_2	9–11	11–13	I	**ADDRESS:** Used to select a register from the task register file.
DB_0–DB_7	12–19	14–16 18–22	I/O	**DATA BUS:** Tri-state, bi-directional 8-bit Data Bus with control determined by BCS. When BCS is high the microprocessor has full control of the data bus for reading and writing the Task Register File. When BCS is low the 82064 controls the data bus to transfer data to or from the buffer.
V_{SS}	20	23		Ground
WR DATA	21	24	O	**WRITE DATA:** Output that shifts out MFM data at a rate determined by Write Clock. Requires an external D flip-flop clocked at 10 MHz. The output has an active pullup and pulldown that can sink 4.8 mA.
$\overline{LATE}$	22	25	O	**LATE:** Output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders.
$\overline{EARLY}$	23	26	O	**EARLY:** Output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders.
WR GATE	24	27	O	**WRITE GATE:** High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.
WR CLOCK	25	29	I	**WRITE CLOCK:** Clock input used to derive the write data rate. Frequency = 5 MHz for the ST506 interface.

3.0 PIN DESCRIPTIONS (Continued)

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
DIR	26	30	O	**DIRECTION:** High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82064's internal comparison of actual cylinder location vs. desired cylinder.
STEP	27	31	O	**STEP:** This signal is used to move the drive head to another cylinder at a programmable frequency. Pulse width = 1.6 μs for a step rate of 3.2 μs/step, and 8.4 μs for all other step rates.
DRDY	28	32	I	**DRIVE READY:** If DRDY from the drive goes low, the command will be terminated.
INDEX	29	33	I	**INDEX:** Signal from the drive indicating the beginning of a track. It is used by the 82064 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.
WR FAULT	30	34	I	**WRITE FAULT:** An error input to the 82064 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.
TRACK 000	31	35	I	**TRACK ZERO:** Signal from the drive which indicates that the head is at the outermost cylinder. Used to verify proper completion of a RESTORE command.
SC	32	36	I	**SEEK COMPLETE:** Signal from the drive indicating to the 82064 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.
RWC	33	37	O	**REDUCED WRITE CURRENT:** Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.
DRUN	34	38	I	**DATA RUN:** This signal informs the 82064 when a field of all ones or all zeroes has been detected in the read data stream by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods.
BRDY	35	39	I	**BUFFER READY:** Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host μP. Only the rising edge indicates the condition.
BDRQ	36	40	O	**BUFFER DATA REQUEST:** Activated during Read or Write commands when a data transfer between the host and the 82064's sector buffer is required. Typically used as a DMA request line.
RD DATA	37	41	I	**READ DATA:** Single ended input that accepts MFM data from the drive.
RD GATE	38	42	O	**READ GATE:** Output that is asserted when a search for an address mark is initiated. It remains asserted until the end of the ID or data field.
RD CLOCK	39	43	I	**READ CLOCK:** Clock input derived from the external data recovery circuits.
V_{CC}	40	44	I	**D.C. POWER:** +5V.
NC	—	5, 6, 17, 28		**NO CONNECTS**

4.0 TASK REGISTER FILE

The Task Register File is a bank of registers used to hold parameter information pertaining to each command. These registers and their addresses are:

A2	A1	A0	READ	WRITE
0	0	0	(Bus Tri-Stated)	(Bus Tri-Stated)
0	0	1	Error Flags	Reduce Write Current
0	1	0	Sector Count	Sector Count
0	1	1	Sector Number	Sector Number
1	0	0	Cylinder Low	Cylinder Low
1	0	1	Cylinder High	Cylinder High
1	1	0	SDH	SDH
1	1	1	Status Register	Command Register

NOTE:
Registers are not cleared by RESET

4.1 Error Register

This read-only register contains specific error status after the completion of a command. If any bit in this register is set, then the Error bit in the Status Register will also be set. The bits are defined as follows:

7	6	5	4	3	2	1	0
BBD	CRC/ECC	O	ID	O	AC	TK000	DM

Bit 7 - Bad Block Detect (BBD)

This bit is set when an ID field has been encountered that contains a bad block mark. The bad block bit is set only during formatting. The 82064 will terminate a command if an attempt is made to read a sector that contains this bit.

Bit 6 - CRC/ECC Data Field Error (CRC/ECC)

When in the CRC mode (SDH register, bit 7 = 0), this bit is set when a CRC error occurs in the data field. When retries are enabled, ten more attempts are made to read the sector correctly. If none of these attempts are successful bit 0 in the STATUS register is also set. If one of the attempts is successful, the CRC/ECC error bit remains set to inform the host that a marginal condition exists; however, bit 0 in the STATUS register is not set.

When in the ECC mode (SDH register, bit 7 = 1), this bit is set when the first non-zero syndrome is detected. When retries are enabled, up to ten attempts are made to correct the error. If the error is successfully corrected, this bit remains set; however, bit 2 of the STATUS register is also set to inform the host that the error has been corrected. If the error is not correctable, the CRC/ECC error bit remains set and bit 0 of the STATUS register is also set.

The data may be read even if uncorrectable errorS exist.

NOTE:
If the long mode (L) bit is set in the READ or WRITE command, no error checking is performed.

Bit 5 - Reserved.

Not used. Set to zero.

Bit 4 - ID Not Found

This bit is set to indicate that the correct cylinder, head, sector, or size parameter could not be found, or that a CRC error occurred in the ID field. Ths bit is set on the first failure and remains set even if the error is recovered on a retry. When recovery is unsuccessful, the Error bit (bit 0) of the STATUS register is also set.

For a SCAN ID command with retries enabled (T = 0), the Error bit in the STATUS register is set after ten unsuccessful attempts have been made to find the correct ID. With retries disabled (T = 1), only two attempts are made before setting the Error bit.

For a READ or WRITE command with retries enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an autoscan and auto-seek are performed. Then ten more retries are made before setting the Error bit. When retries are disabled (T = 1), only two tries are made. No autoscan or auto-seek operations are performed.

Bit 3 - Reserved.

Not used. Set to zero.

Bit 2 - Aborted Command

This bit is set if a command was issued or in progress while DRDY (Pin 28) was deasserted or WR FAULT (Pin 30) was asserted. The Aborted Command bit will also be set if an undefined command is written into the COMMAND register, but an implied seek will be executed.

Bit 1 - TRACK 000 Error (TK000)

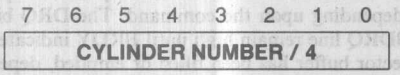
This bit is set only by the RESTORE command. It indicates that TRACK 000 (Pin 31) has not gone active after the issuance of 2048 stepping pulses.

Bit 0 - Data Address Mark

This bit is set during a READ SECTOR command if the Data Address Mark is not found after the proper Sector ID is read.

4.2 Reduce Write Current Register

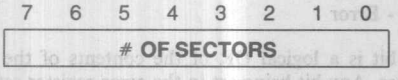
This register is used to define the cylinder number where RWC (Pin 33) is asserted:



The value (0–255) written into this register is internally multiplied by 4 to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to activate on cylinder 4, 02H on cylinder 8 and so on. RWC will be asserted when the present cylinder is greater than or equal to the cylinder indicated by this register. For example, one ST506 compatible drive requires precompensation on cylinder 128 (80H) and above. Therefore the REDUCE WRITE CURRENT register should be loaded with 32 (20H). A value of FFH will keep the RWC output inactive regardless of the actual cylinder number.

4.3 Sector Count Register

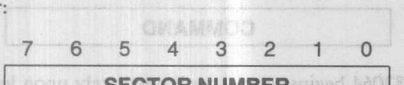
This register is used to define the number of sectors that need to be transferred to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command.



The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a 1 sector transfer, etc. This register is ignored when single sector commands are specified in the Command register.

4.4 Sector Number

This register holds the sector number of the desired sector:

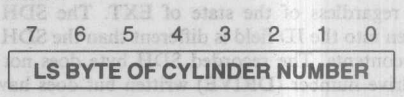


For a multiple sector command it specifies the first sector to be transferred. It is decremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255. The ID Not Found bit will be set if the desired sector cannot be located on the track.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formatting a disk. See the WRITE FORMAT command description for further explanation.

4.5 Cylinder Number Low Register

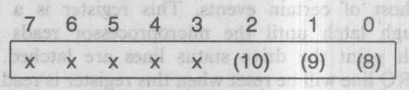
This register holds the lower byte of the desired cylinder number:



It is used in conjunction with the CYLINDER NUMBER HIGH register to specify a range of 0 to 2048 tracks.

4.6 Cylinder Number High Register

This register holds the three most significant bits of the desired cylinder number:



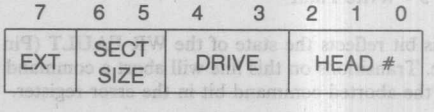
x = ignored

The 82064 contains a pair of registers that store the actual position where the R/W head are located. The CYLINDER NUMBER HIGH and LOW registers are considered the cylinder destination registers for seeks and other commands. The 82064 compares its internal registers to the destination registers and issues the number of steps in the right direction to make both sets of registers equal. After a command is executed, the internal cylinder position registers' contents are equal to the cylinder high/low registers. If a drive number change is detected on a new command, the 82064 automatically reads an ID field to update its internal cylinder position registers. This affects all commands except a RESTORE.

When a RESTORE command is executed, the internal head location registers are reset to zero while DIR and STEP move the heads to track zero.

4.7 Sector/Drive/Head (SDH) Register

The SDH register contains the desired sector size, drive number, and head number parameters. The format is shown below.



Both head number and sector size are compared against the disk's ID field. Head select and drive select lines are not available as outputs from the 82064 and must be generated externally.

Bit 7, the extension bit (EXT), is used to select between the CRC or ECC mode. When bit 7 = 1, the ECC mode is selected for the data field. When bit 7 = 0, the CRC mode is selected. The CRC is checked on the ID field regardless of the state of EXT. The SDH byte written into the ID field is different than the SDH Register contents. The recorded SDH byte does not have the drive number (DRIVE) written but does have the BAD BLOCK mark written.

Note that use of the extension bit requires the gap lengths to be modified as described in the WRITE FORMAT command description.

4.8 Status Register

The status register is a read-only register which informs the host of certain events. This register is a flow-through latch until the microprocessor reads it at which point the drive status lines are latched. The INTRQ line will be reset when this register is read. The format is:

7	6	5	4	3	2	1	0
BUSY	READY	WF	SC	DRQ	DWC	CIP	ERROR

Bit 7 - Busy

This bit is asserted when a command is written into the COMMAND register and, except for the READ command, is deasserted at the end of the command. When executing a READ command, Busy will be deasserted when the sector buffer is full. Commands should not be loaded into the COMMAND register when Busy is set. When the Busy bit is set, no other bits in the STATUS or ERROR registers are valid.

During other non-data transfer commands, Busy should be ignored as it will go active for short periods.

Bit 6 - Ready

This bit reflects the state of the DRDY (Pin 28) line at the time the microprocessor reads the status register. Transitions on the DRDY line will abort a command and set the aborted command bit in the error register.

Bit 5 - Write Fault

This bit reflects the state of the WR FAULT (Pin 30) line. Transitions on this line will abort a command and set the aborted command bit in the error register.

Bit 4 - Seek Complete

This bit reflects the state of the SC (Pin 32) line. Commands which initiate a seek will pause until Seek Complete is set. This bit is latched after an aborted command error.

Bit 3 - Data Request

The Data request bit (DRQ) reflects the state of the BDRQ (Pin 36) line. It is set when the sector buffer should be loaded with data or read by the host processor, depending upon the command. The DRQ bit and the BDRQ line remain high until BRDY indicates that the sector buffer has been filled or emptied, depending upon the command. BRDQ can be used for DMA.

Bit 2 - Data Was Corrected (DWC)

When set, this bit indicates that an ECC error has been detected during a read operation, and that the data in the sector buffer has been corrected. This provides the user with an indication that there may be a marginal condition within the drive before the errors become uncorrectable. This bit is forced to zero when not in the ECC mode.

Bit 1 - Command in Progress

When this bit is set, a command is being executed and a new command should not be loaded until it is cleared. Although a command may be executing, the sector buffer is still available for access by the host processor. When the 82064 is no longer busy (bit 7 = 0) the status register can be read. If CIP is set, only the status register can be read regardless of which register is selected.

Bit 0 - Error

This bit is a logical OR of the contents of the error register. Any bit being set in the error register sets this bit. The host must read the ERROR register to determine what type of error occurred. This bit is cleared when a new command is loaded.

4.9 Command Register

This write-only register is loaded with the desired command:

7	6	5	4	3	2	1	0
COMMAND							

The 82064 begins to execute immediately upon loading any value into this register. This register should not be written while the Busy or Command in Progress bits are set in the STATUS register. The INTRQ line (Pin 3) if set, will be cleared by a write to the COMMAND register.

Instruction Set

The 82064 WDC instruction set contains six commands. Prior to loading the command register, the host processor must first set up the Task Register File with the information needed for the command. Except for

the COMMAND register, the registers may be loaded in any order. If a command is in progress, a subsequent write to the COMMAND register will be ignored. A command is finished when the command in progress (CIP) bit in the STATUS register is cleared. See the Command Section for an explanation of each command.

COMMAND	7	6	5	4	3	2	1	
RESTORE	0	0	0	1	R3	R2	R1	R0
SEEK	0	1	1	1	R3	R2	R1	R0
READ SECTOR	0	0	1	0	I	M	0	T
WRITE SECTOR	0	0	1	1	0	M	0	T
SCAN ID	0	1	0	0	0	0	0	T
WRITE FORMAT	0	1	0	1	0	0	0	0
COMPUTE CORRECTION	0	0	0	0	1	0	0	0
SET PARAMETER	0	0	0	0	0	0	0	S

R 3–0 = Rate Field

For 5 MHz WR Clock:

0000 — ≈35 μs
0001 — 0.5 ms
0010 — 1.0 ms
0011 — 1.5 ms
0100 — 2.0 ms
0101 — 2.5 ms
0110 — 3.0 ms
0111 — 3.5 ms
1000 — 4.0 ms
1001 — 4.5 ms
1010 — 5.0 ms
1011 — 5.5 ms
1100 — 6.0 ms
1101 — 6.5 ms
1110 — 3.2 μs
1111 — 16 μs

COMMAND 7 6 5 4 3 2 1
T = Retry Enable
T = 0 Enable Retries T = 1 Disable Retries
M = Multiple Sector Flag
M = 0 Transfer 1 Sector M = 1 Transfer Multiple Sectors
I = Interrupt Enable
I = 0 Interrupt at BDRQ time I = 1 Interrupt at end of command
S = Error Correction Span
S = 0 5-bit Span S = 1 11-bit Span

5.0 PROGRAMMING THE 82064

This section consists of two parts. The first part gives an explanation of each command, a flowchart showing the 82064's sequence of events, and the commands' sequence of events as seen by the host microprocessor. The second section shows flowcharts of general software routines and their PLM equivalent, for both polled and interrupt driven software.

The designer must remember that the 82064 expects a full sector buffer that can be isolated from the host during data transfers between the 82064 and the disk. Since the 82064 assumes a full sector buffer is available, it does not check for data overrun or underrun error conditions. If such a condition occurs, corruption of data will happen and the host will have no indication of an error. The design must guarantee against over-run and under-run conditions when not using the sector buffer approach.

5.1 Commands

A command is placed into the command register only after the Task Registers have been written with proper values. The Task Registers may be loaded in any order. A command, once started, can only be terminated by a hardware reset to the 82064. This may corrupt data on the disk by removing necessary control signals out of sequence.

The general sequence of a command is as follows:

— The host loads the Task Registers
— The host loads the Command Register
— The 82064 locates the correct cylinder
— Data transfer takes place
— The 82064 issues an interrupt

Restore Command – 0 0 0 1 R3 R2 R1 R0

The Restore command is used to position the heads to track 0. This command is usually issued to the 82064 on power-up to initialize internal registers. The user specified rate field (R3–R0) is stored internally for FUTURE use in commands with implied seeks.

The step rate value is not used with this command. The actual stepping rate used is dependent upon the handshake delay between the 82064 issuing a step pulse and the drive returning a seek complete for each track. After each step pulse is issued, the 82064 waits for a rising edge on the Seek Complete (SC) line before issuing the next pulse. If 8 index pulses are received without a rising edge on SC, the 82064 will switch to sampling the level of the SC line. If after 2048 step pulses the Track 00 signal has not gone active, the 82064 will terminate

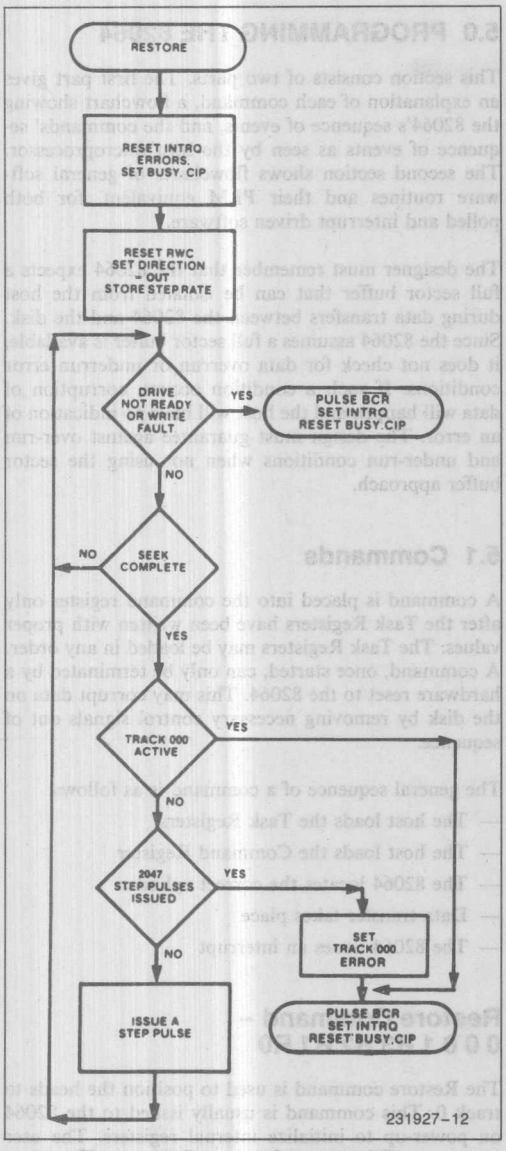


Figure 13. Restore Command Flow

the command, assert INTRQ and set the TRACK 000 bit in the Error Register. The command will terminate if WR Fault goes active or DRDY goes inactive at any time. Figure 13 is a flow chart of the command.

Seek Command – 0 1 1 1 R3 R2 R1 R0

The Seek command positions the heads to the cylinder specified in the Task Registers. The direction and number of step pulses issued is calculated by comparing the cylinder high/low registers to an internal "present position" cylinder register. The present position register is updated after all step pulses are issued and the command is terminated.

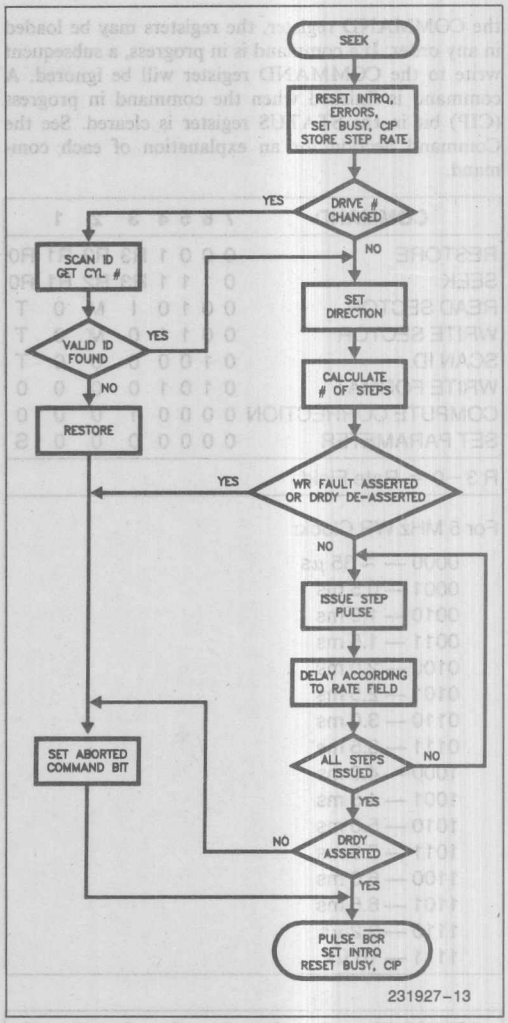


Figure 14. Seek Command Flow

The actual stepping rate is taken from the rate field bits (R3–R0) and stored for future use. The command terminates at once if WR FAULT goes active or DRDY goes inactive at any time. Figure 14 is a flowchart of the command.

Since the data transfer commands feature implied seeks, this command is of use mainly to those using multiple drives and software that can take advantage of overlapped seeks.

Scan ID Command – 0 1 0 0 0 0 0 T

The Scan ID command is used by both the 82064 and the host to update the SDH, the Sector Number, Cylinder and internal present position registers. Once the command is issued, the Seek Complete line is sampled until valid. The first ID field found, as indicated by the address mark, is loaded into the previously mentioned registers. The Bad Block bit will be set if detected, and the command will terminate. ID CRC errors will start the search sequence over for a maximum of 10 index pulses, but the registers will be loaded with whatever data the 82064 had perceived as ID information. Improper states on WR Fault on DRDY will terminate the command. Figure 15 is the flow chart of the command.

The main use for this command is to determine where the heads are currently located and what size the sectors are (i.e. 256, 512 etc.). Without this command, it would be necessary to recall the heads to track zero and then step out to the desired cylinder each time a drive was changed. Specifying the wrong sector size would yield an ID not found error. This command enables the system to read the disk drive to determine what size sectors were recorded.

Read Sector Command – 0 0 1 0 I M 0 T

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the READ SECTOR command, the 82064 checks the CYLINDER NUMBER LOW/HIGH register pair against an internal cylinder position register to see if they are equal. If not, the direction and number of steps are calculated and a seek takes place. If an implied seek is performed, the 82064 will search until a rising edge of SC is received. The WR FAULT and DRDY lines are monitored throughout the command.

Once the Seek Complete (SC) line is high (with or without an implied seek having occurred), the search for an ID field begins. If T = 0 (retries enabled), the 82064 must find an ID with the correct cylinder number, head, sector size, and CRC within 10 revolutions, or a Scan ID and re-Seek will be performed. The search for the proper ID will again be tried for up to 10 revolutions. If the correct sector is still not found, the appropriate error bits will be set and the command terminated. Data CRC errors will also be retried for up to 10 revolutions (if T = 0).

If T = 1 (retries disabled), the ID search must find the correct sector within 2 revolutions or the appropriate error bits will be set and the command terminated.

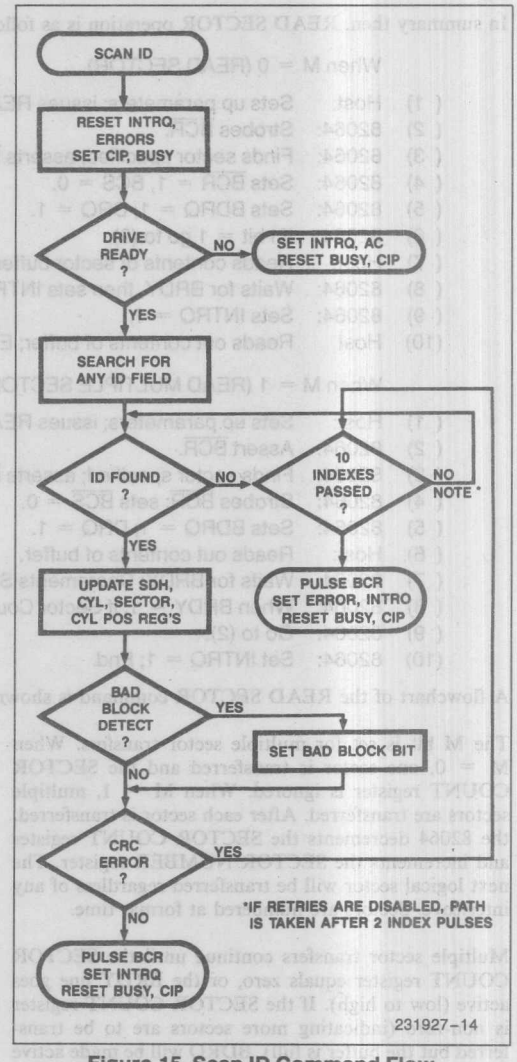


Figure 15. Scan ID Command Flow

Both the READ SECTOR and WRITE SECTOR commands feature a "simulated completion" to ease programming. DRQ/BDRQ will be generated upon detecting an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

When the data address mark is found, the 82064 is ready to tranfer data to the sector buffer. After the data has been transferred, the I bit is checked. If I = 0, INTRQ is made active coincident with BDRQ, indicating that a transfer of data from the buffer to the host processor is required. If I = 1, INTRQ will occur at the end of the command, i.e. after the buffer is unloaded by the host.

In summary then, READ SECTOR operation is as follows:

When M = 0 (READ SECTOR)

(1)	Host:	Sets up parameters; issues READ SECTOR command.
(2)	82064:	Strobes $\overline{BCR}$.
(3)	82064:	Finds sector specified; asserts $\overline{BCR}$ and $\overline{BCS}$; transfers data to buffer.
(4)	82064:	Sets $\overline{BCR}$ = 1, $\overline{BCS}$ = 0.
(5)	82064:	Sets BDRQ = 1; DRQ = 1.
(6)	82064:	If I bit = 1 go to (9).
(7)	Host:	Reads contents of sector buffer.
(8)	82064:	Waits for BRDY, then sets INTRQ = 1: END.
(9)	82064:	Sets INTRQ = 1.
(10)	Host	Reads out contents of buffer; END.

When M = 1 (READ MULTIPLE SECTOR)

(1)	Host:	Sets up parameters; issues READ SECTOR command.
(2)	82064:	Assert $\overline{BCR}$.
(3)	82064:	Finds sector specified; asserts $\overline{BCR}$ and $\overline{BCS}$; transfers data to buffer.
(4)	82064:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(5)	82064:	Sets BDRQ = 1; DRQ = 1.
(6)	Host:	Reads out contents of buffer.
(7)	82064:	Waits for BRDY; Decrements SECTOR COUNT; increments SECTOR NUMBER.
(8)	82064:	When BRDY = 1, if Sector Count = 0 then go to (10).
(9)	82064:	Go to (2).
(10)	82064:	Set INTRQ = 1; End.

A flowchart of the READ SECTOR command is shown in Figures 16A and 16B.

The M bit is set for multiple sector transfers. When M = 0, one sector is transferred and the SECTOR COUNT register is ignored. When M = 1, multiple sectors are transferred. After each sector is transferred, the 82064 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector will be transferred regardless of any interleave. Sectors are numbered at format time.

Multiple sector transfers continue until the SECTOR COUNT register equals zero, or the BRDY line goes active (low to high). If the SECTOR COUNT register is non-zero (indicating more sectors are to be transferred but the buffer is full), BDRQ will be made active and the host must unload the buffer. After this occurs, the buffer will again be free to accept the remaining sectors from the 82064. This scheme enables the user to transfer more sectors than the buffer memory has capacity for.

Write Sector Command – 0 1 1 1 0 M 0 T

The WRITE SECTOR command is used to write one or more sectors of data to the disk from the sector buffer. Upon receipt of a WRITE SECTOR command the 82064 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. The WR FAULT and DRDY lines are checked throughout the command.

When the Seek Complete (SC) line is found to be true (with or without an implied seek having occurred), the BDRQ signal is made active and the host proceeds to load the buffer. Once BRDY goes high, the ID field with the specified cylinder number, head, and sector size is searched for. Once found, WR GATE is made active and the data is written to the disk. If retries are enabled (T = 0), and if the ID field cannot be found within 10 revolutions, a Scan ID and re-Seek are performed. If the correct ID field is not found within 10 additional revolutions, the ID Not Found error bit is set and the command is terminated. If retries are disabled, (T = 1) and if the ID field cannot be found within 2 revolutions, the ID Not Found error bit is set and the command is terminated.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is decremented and the SECTOR COUNT register is incremented after the transfer to the disk takes place. During multiple sector transfers if BRDY is asserted after the first sector is transferred from the buffer, the 82064 will transfer the next sector before issuing BDRQ. The 82064 will set BDRQ and wait for the host processor to place more data in the buffer.

In summary then, the WRITE SECTOR operation is as follows:

When M = 0, 1 (WRITE SECTOR)

(1)	Host:	Sets up parameters; issues WRITE SECTOR command.
(2)	82064:	Sets BDRQ = 1, DRQ = 1.
(3)	Host:	Loads sector buffer with data.
(4)	82064:	Waits for BRDY = 0 to 1.
(5)	82064:	Finds specified ID field; writes sector to disk.
(6)	82064:	If M = 0, then set INTRQ = 1; END.
(7)	82064:	Increment SECTOR NUMBER register; decrement SECTOR COUNT register.
(8)	82064:	If SECTOR = 0, then set INTRQ = 1; END.
(9)	82064:	Go to (2).

A flowchart of the WRITE SECTOR command is shown in Figure 17.

Write Format Command 01010000

The WRITE FORMAT command is used to format one track using the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of sector data. Shown in Figure 18 is the contents of the sector buffer for a 32 sector/track format with an interleave factor of two. Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. A 00 Hex is normal; an 80H indicates a bad block mark for the sector. In the example of Figure 18, sector 04 will get a back block mark recorded. Any attempt to access sector 4 in the future will terminate the command.

The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may contain any value. Its only purpose is to generate a BRDY to tell the 82064 to begin formatting the track. An implied seek is in effect on this command. As for other commands, if the drive number has been changed an ID field will be scanned for cylinder position information before the implied seek is performed. If no ID field can be read (because the track had been erased or because an incomplete format had been used), an ID Not Found error will result and the WRITE FORMAT command will be aborted. This can be avoided by issuing a RESTORE command before formatting.

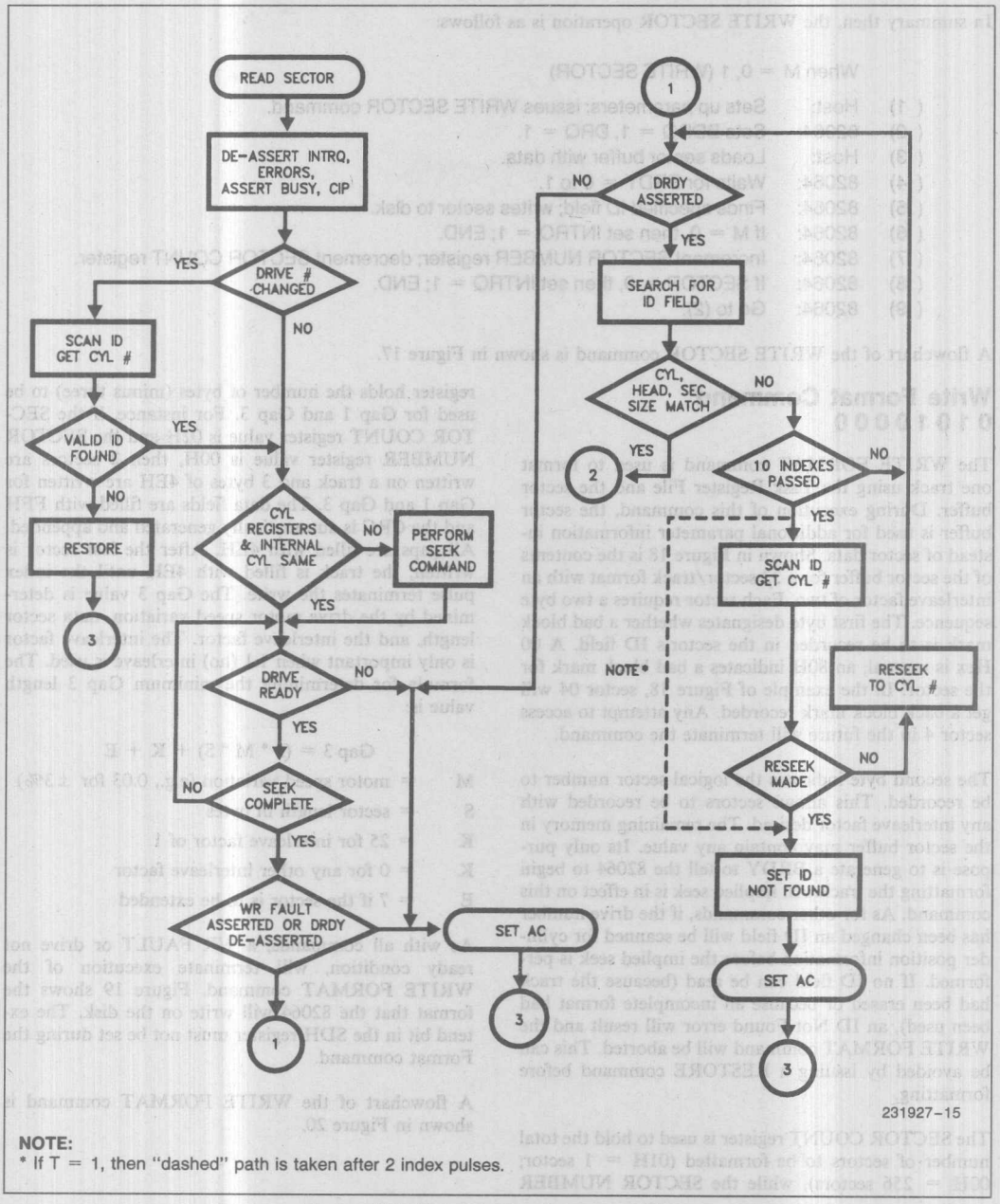
The SECTOR COUNT register is used to hold the total number of sectors to be formatted (01H = 1 sector; 00H = 256 sectors), while the SECTOR NUMBER register holds the number of bytes (minus three) to be used for Gap 1 and Gap 3. For instance, if the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are written on a track and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC is automatically generated and appended. All gaps are filled with 4EH. After the last sector is written, the track is filled with 4EH until the index pulse terminates the write. The Gap 3 value is determined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 (no) interleave is used. The formula for determining the minimum Gap 3 length value is:

$$\text{Gap 3} = (2 * M * S) + K + E$$

M	=	motor speed variation (e.g., 0.03 for ±3%)
S	=	sector length in bytes
K	=	25 for interleave factor of 1
K	=	0 for any other interleave factor
E	=	7 if the sector is to be extended

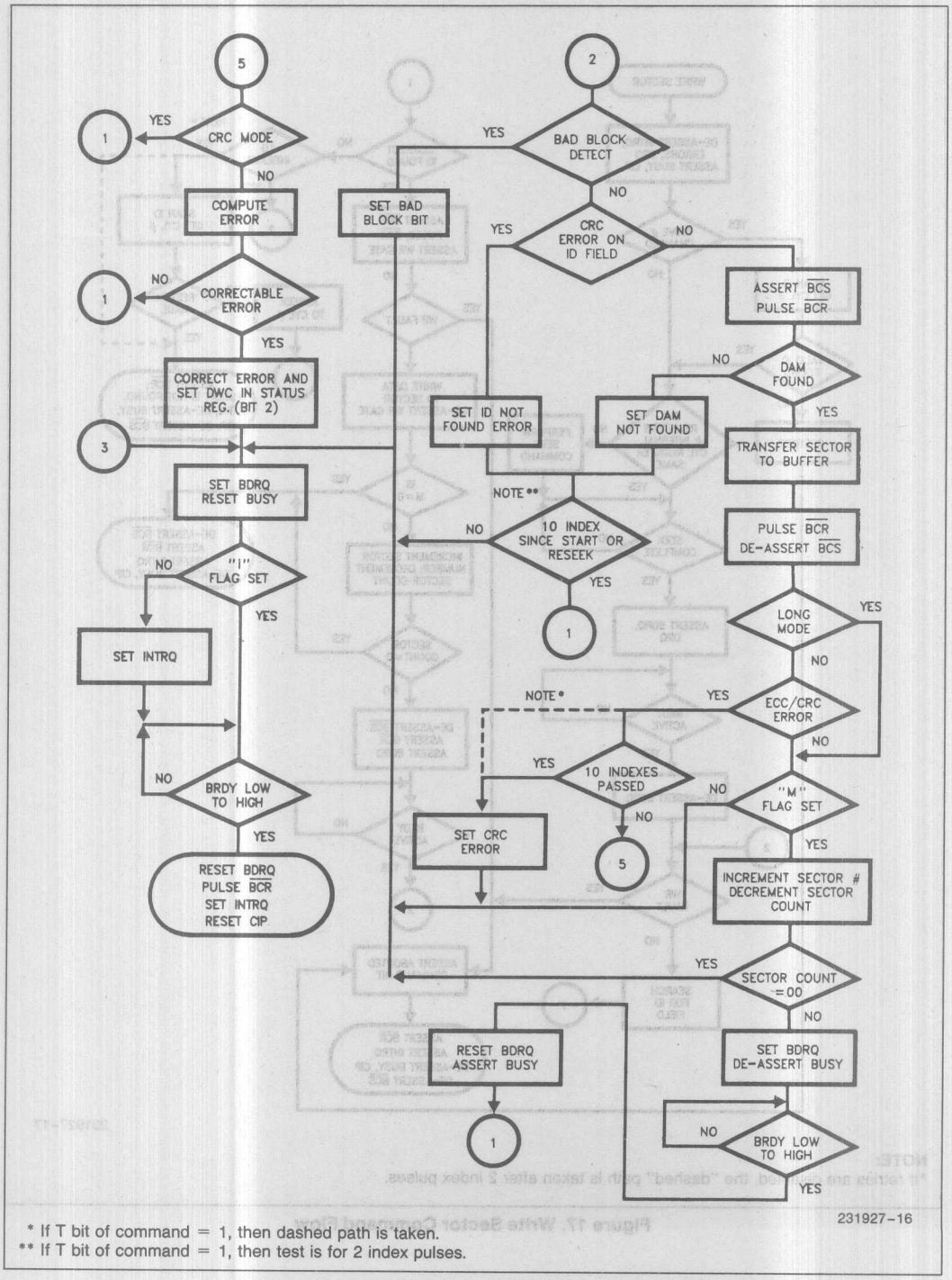
As with all commands, a WR FAULT or drive not ready condition, will terminate execution of the WRITE FORMAT command. Figure 19 shows the format that the 82064 will write on the disk. The extend bit in the SDH register must not be set during the Format command.

A flowchart of the WRITE FORMAT command is shown in Figure 20.



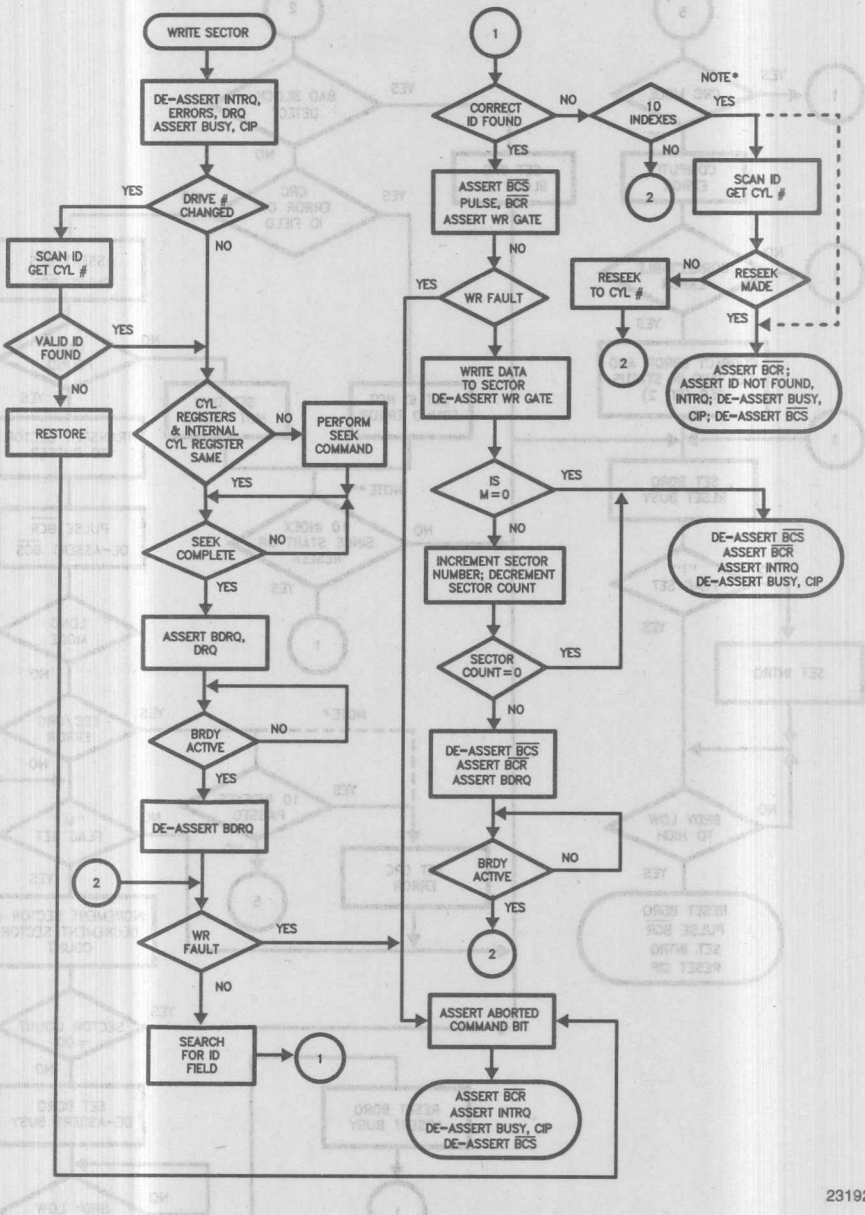
NOTE:
* If T = 1, then "dashed" path is taken after 2 index pulses.

Figure 16A. Read Sector Command Flow



* If T bit of command = 1, then dashed path is taken.
** If T bit of command = 1, then test is for 2 index pulses.

Figure 16B. Read Sector Command Flow (Continued)



NOTE:
*If retries are disabled, the "dashed" path is taken after 2 index pulses.

Figure 17. Write Sector Command Flow

00	00	00	10	00	01	00	11	00	02	00	12	00	03	00	13
80	04	00	14	00	05	00	15	00	06	00	15	00	07	00	17
00	08	00	18	00	09	00	19	00	0A	00	19	00	0B	00	1B
00	0C	00	1C	00	0D	00	1D	00	0E	00	1E	00	0F	00	1F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 18. Sector Buffer Contents For Format

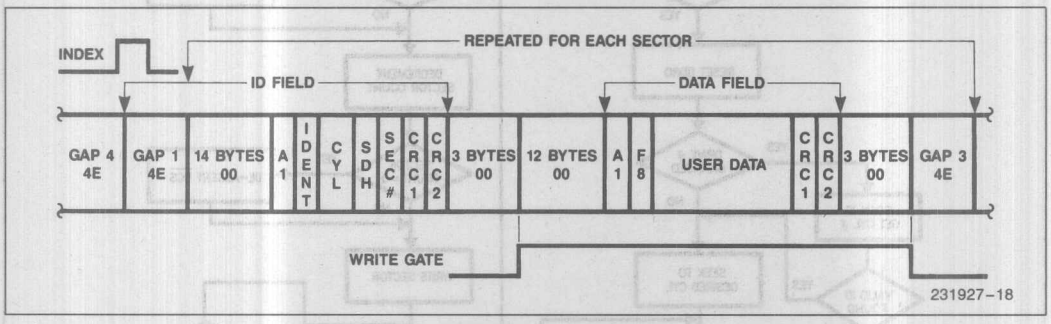


Figure 19. 82064 Sector Format

5.2 Software Section: General Programming

This section describes the software needed to communicate with the 82064 in order to store and retrieve data. This chapter describes the software in a general manner and Appendix B contains the actual implementation used to exercise the 82064 SBX board.

Polled Mode

As discussed in the Polled Interface Section, the 82064 does not directly support polled operation for data transfers without the addition of hardware. This section is based upon the polled interface as described in the Polled Interface Section.

The six 82064 commands can be divided into two groups, those with data transfers and those without. The commands that do not use the sector buffer are: Restore, Seek and Scan ID. The functions of each command are explained in the Commands Section. Figure 21 is a flowchart of a polled operation and a PLM example.

The last status that was read will contain any error conditions that might have occurred during the command.

For commands that do make use of the sector buffer, the size of the sector buffer will affect the software. If the sector buffer is equal in size to one sector, then a carry out of an address counter (for the sector buffer) as the buffer is being filled will indicate to the 82064 that the command should continue. If the sector buffer size is equal to two or more disk sectors, and only one sector is being transferred, then the carry out signal would not go active, and the 82064 will be forever waiting for BRDY. In this case an I/O port would have to be used to generate this signal for the 82064 so that command execution can finish. Figure 22 is a flowchart of the READ SECTOR command, and its PLM representation. The WRITE SECTOR and FORMAT TRACK commands are equivalent in terms of software interfacing. Their flowcharts and their PLM equivalents are shown in Figure 23.

Once the command register is written the 82064 requests a data transfer before locating the proper track. Once the buffer is filled and BRDY is asserted, the 82064 will locate the target track and sector. If the ID is not located before the selected number of retries have occurred, the 82064 will terminate the command. The data transferred to the sector buffer will not have been used. Once the command has finished (i.e., CIP = 0), the status and error registers will inform the host of an error.

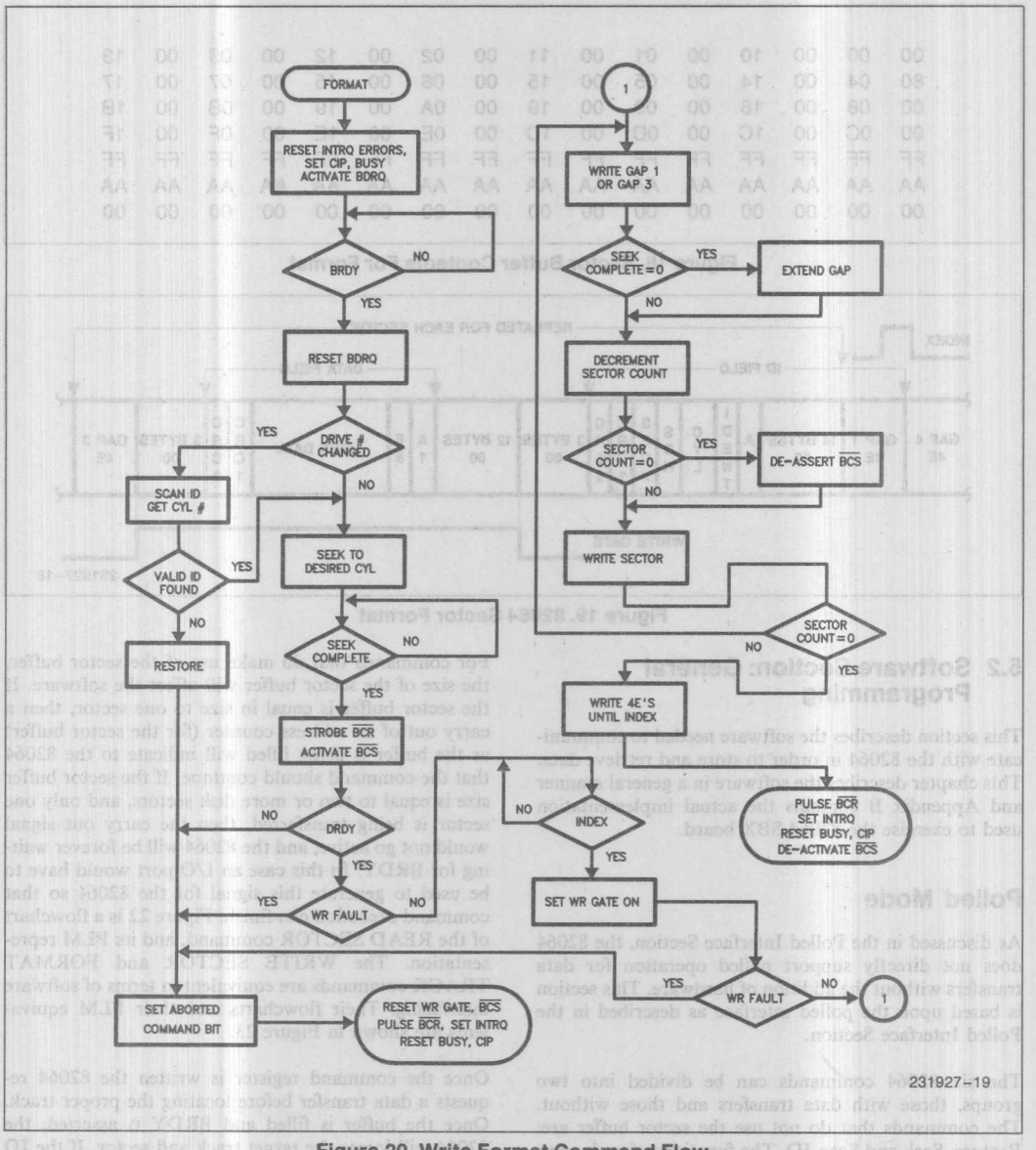
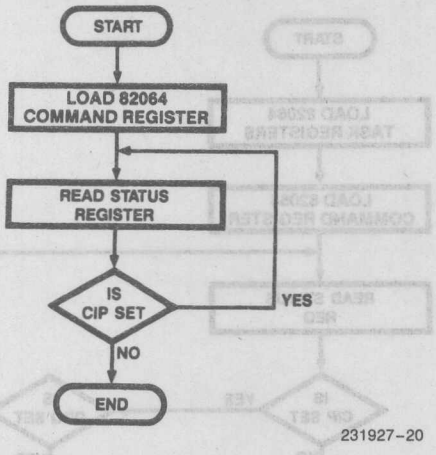
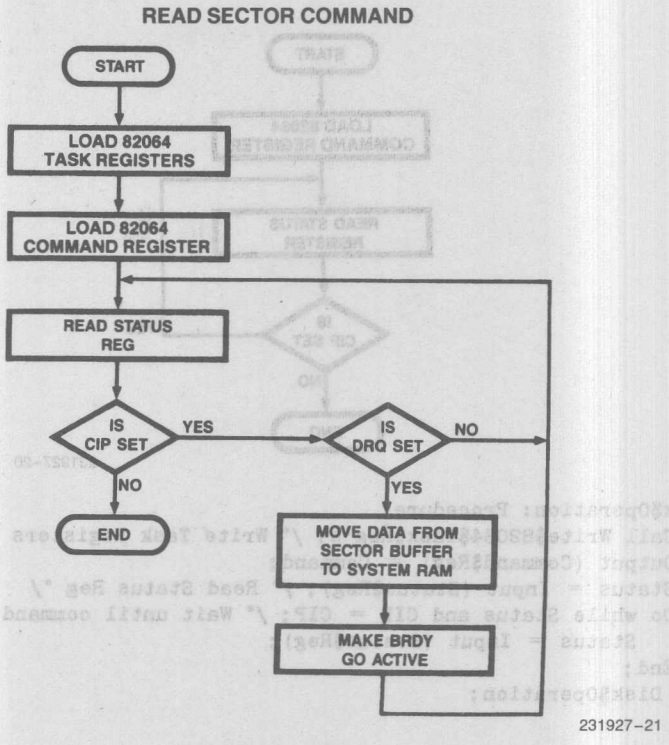


Figure 20. Write Format Command Flow



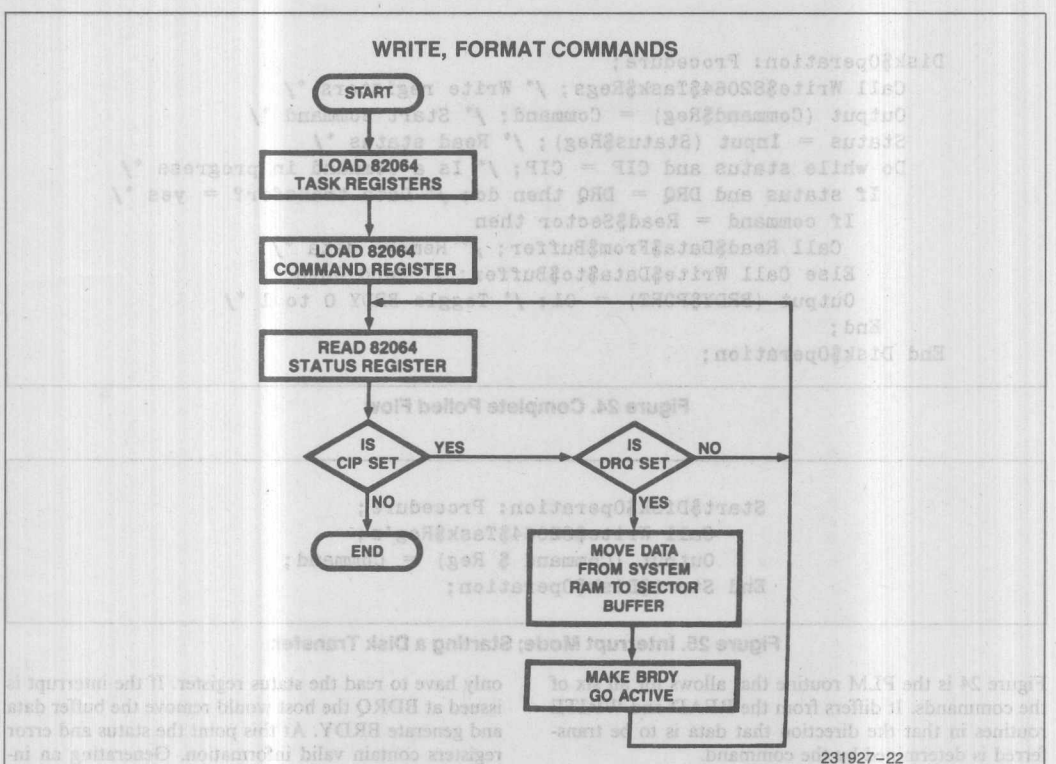

```
Disk$Operation: Procedure;
   Call Write$82064$Task$Reg's; /* Write Task Registers */
   Output (Command$Reg) = Command;
   Status = Input (Status$Reg); /* Read Status Reg */
   Do while Status and CIP = CIP; /* Wait until command finishes */
      Status = Input (Status$Reg);
   End;
End Disk$Operation;
```

Figure 21. Polling Status




```
Disk$Operation: Procedure;
    Call Write$82064$Task$Regs;
    Output (Command $ Reg) = Command;
    Status = Input (Status$Reg);
    Do while Status and CIP = CIP;
        If Status and DRQ = DRQ then Do;
          Call Read$Data$From$Buffer;
          Output (BRDY$PORT) = 01;
        End;
        Status = Input (Status$Port)
    End;
End Disk$Operation;
```

Figure 22. Polling For Read Data




```
Disk$Operation: Procedure;
  Call Write$82064$Task$Regs;
  Output (Command$Reg) = Command;
  Status = Input (Status$Reg);
  Do while status and CIP = CIP;
    If status and DRQ = DRW then do;
      Call Write$Data$to$Buffer;
      Output (BRDY$Port) = 01; /* Make BRDY go active */
    End;
    Status = Input (Status$Reg)
  End;
End Disk$Operation;
```

Figure 23. Polling For Write Data

```
Disk$Operation: Procedure;
   Call Write$82064$Task$Regs; /* Write registers */
   Output (Command$Reg) = Command; /* Start command */
   Status = Input (Status$Reg); /* Read status */
   Do while status and CIP = CIP; /* Is a command in progress */
     If status and DRQ = DRQ then do; /* Data transfer? = yes */
       If command = Read$Sector then
        Call Read$Data$From$Buffer; /* Remove data */
       Else Call Write$Data$to$Buffer; /* Send data */
       Output (BRDY$PORT) = 01; /* Toggle BRDY 0 to 1 */
     End;
End Disk$Operation;
```

Figure 24. Complete Polled Flow

```
Start$Disk$Operation: Procedure;
   Call Write$82064$Task$Reg's;
   Output (Command $ Reg) = Command;
End Start$Disk$Operation;
```

Figure 25. Interrupt Mode; Starting a Disk Transfer

Figure 24 is the PLM routine that allows for all six of the commands. It differs from the READ and WRITE routines in that the direction that data is to be transferred is determined by the command.

Figure 24 also works for multiple sector transfers. However, the BRDY signal must be generated in hardware (the carry-out of an address counter).

Interrupt Mode

Interrupt driven software is chosen when the microprocessor must execute other tasks and cannot sit waiting for the disk to reposition its heads, as in a polled environment. The delay in repositioning heads can be anything from a couple of milliseconds to a second or more.

The 82064's interrupt (INTRQ) pin goes active to indicate that the command has finished. The READ SECTOR command provides the programmable choice of having the interrupt occur at the end of the data transfer or the normal end of the command. The reason for this option is that when the 82064 signals that a data transfer is required (via BDRQ, DRQ) the disk has been read and the data has been placed in the buffer. The host would remove the data and issue BRDY. The 82064 would then issue an interrupt indicating that the command has finished. The interrupt procedure would only have to read the status register. If the interrupt is issued at BDRQ the host would remove the buffer data and generate BRDY. At this point the status and error registers contain valid information. Generating an interrupt at BDRQ time may save some systems some software overhead.

The WRITE SECTOR and FORMAT commands do not have this option because the sector buffer is filled before the track and sector are located. Hence, there can be significant delays between asking for data and the command terminating.

In an interrupt driven environment, the 82064 can interface to a DMA controller for data transfers between the sector buffer and the host's RAM. If a DMA controller is not available an interrupt must be generated via the BDRQ line. However, BDRQ can stay active for long periods of time (until BRDY is generated). The interrupt sensing logic must take this into account to avoid being retriggered constantly. Intel's 8259A Interrupt Controller 8259A provides that capability. It should be programmed for edge triggered interrupts or the end of interrupt byte must not be issued until BDRQ is removed to prevent retriggering.

Figure 25 is a PLM example of starting a disk operation in an interrupt driven environment. The command starts, and some indefinite amount of time later an interrupt would be generated, indicating service is required.

```
End$of$Transfer: Procedure Interrupt;
    Status = Input (Status$Register);
    Output (8259A PIC) = End$of$Interrupt;
End End$of$Transfer;
```

Figure 26. Checking Status via Interrupt

```
Service$Disk$Controller: Procedure Interrupt;
    Status = Input (Status$Port);
    If Status and DRQ = DRQ then
        Call Transfer$Data$To/From$Buffer;/* Enable DMAC */
    Output (8259A PIC) = End$of$Interrupt;
End Service$Disk$Controller;
```

Figure 27. Complete Interrupt Procedure

If a DMA controller is used, it would have to be programmed and initialized before the command is issued to the 82064. Recall that once a data transfer between the microprocessor and 82064 has finished, BRDY must be set high. As long as BRDY is generated from hardware, no microprocessor intervention is needed. If BRDY is generated by an I/O port the microprocessor will have to perform this function (this will be the case with any system that has a sector buffer larger than one sector). (One option could be to generate an interrupt from the terminal count pin of the DMA controller. The microprocessor would then issue a BRDY.) Data transfers between host RAM and the sector buffer would be handled without microprocessor intervention. The interrupt would then signal that the command has finished as shown in Figure 26. The only operation the host processor would perform is to check the status register of the 82064 for any error conditions.

If BDRQ is used to generate an interrupt in addition to the normal interrupt, then the routines shown in Figure 27 will check the status register to see if a data transfer should be executed or if the command is finished. If DRQ is not set, the command has finished and any error conditions would be in the status register.

Another possibility would be to have separate interrupt routines for the two possible sources of interrupts (INTRQ, BRDQ). There would then be no need to test the status to see which interrupt had occurred.

6.0 APPLICATION EXAMPLE

This section shows an application using the 82064 interfaced to the SBX bus. A quick overview of the SBX bus is provided (pin descriptions, general wave forms) as a background for the application. Designing the 82064 onto an SBX Multimodule board was chosen to highlight the size and complexity differences between earlier TTL, MSI, LSI-based disk controller boards and what is possible using the 82064. Both the hardware and software sections will be applicable to most other designs using the 82064. This design example is called SBX82064 and does not represent a real product offered by Intel Corporation. Appendix C contains the schematic of the SBX board.

The advantage of the SBX Multimodule is that it permits the system to be tailored for specific needs with a minimum of effort. The advantage of an SBX based disk controller is that a current system can make use of the capacity, reliability and speed of a hard disk with no (or minimal) hardware redesign.

6.1 iSBX Bus Multimodule Boards

The iSBX Multimodule boards are small, specialized, I/O mapped boards which plug onto base boards. The iSBX boards connect to the iSBX bus connector and convert the iSBX bus signals to a defined I/O interface.

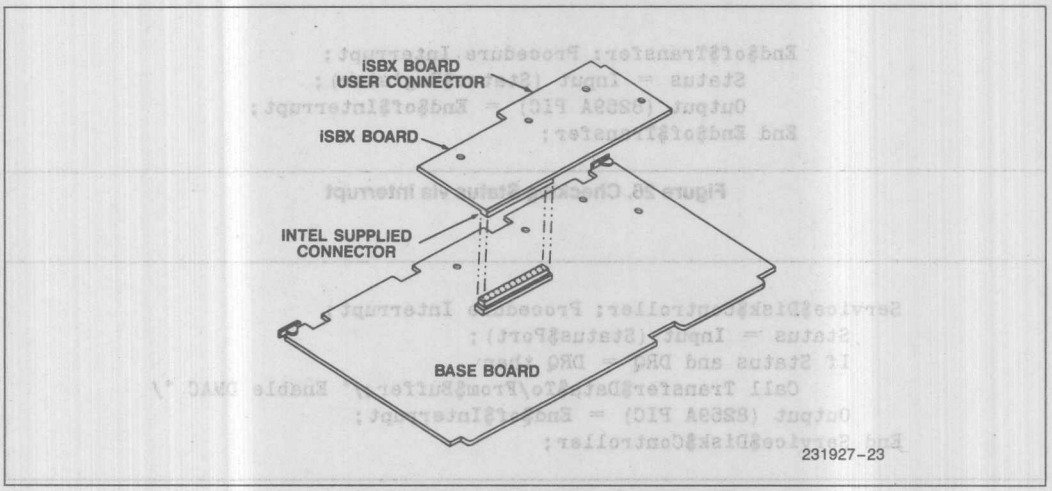


Figure 28. iSBX Multimodule Board Concept (Double Wide)

Base Boards

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. In 8-bit systems, the base board decodes all but the lower three addresses in generating the iSBX Multimodule board chip selects. In 16-bit systems, the base board decodes all but the lower order four addresses in generating the iSBX Multimodule board chip selects. Thus, a base board would normally reserve two blocks of 8 I/O ports for each iSBX socket it provides.

There are two classes of base boards, those with Direct Memory Access (DMA) support and those without. Base boards with DMA support are boards with DMA controllers on them. These boards, in conjunction with an iSBX Multimodule board (with DMA capability), can perform direct I/O to memory or memory to I/O operations.

iSBX Bus Interface

The iSBX bus interface can be grouped into six functional classes:

1. Control Lines
2. Address and Chip Select Lines
3. Data Lines
4. Interrupt Lines
5. Option Lines
6. Power Lines

Control Lines

The following signals are classified as control lines:

COMMANDS:
- $\overline{\text{IORD}}$ (I/O Read)
- $\overline{\text{IOWRT}}$ (I/O Write)

DMA:
- $\overline{\text{MDRQT}}$ (DMA Request)
- $\overline{\text{MDACK}}$ (DMA Acknowledge)
- TDMA (Terminate DMA)

INITIALIZE:
- RESET

CLOCK:
- MCLK (iSBX Multimodule Clock)

SYSTEM CONTROL:
- $\overline{\text{MWAIT}}$
- $\overline{\text{MPST}}$ (iSBX Multimodule Board Present)

Command Lines ($\overline{\text{IORD}}$, $\overline{\text{IOWRT}}$)

The command lines are active low signals which provide the communication link between the base board and the iSBX Multimodule board. An active command line, conditioned by chip select, indicates to the iSBX Multimodule board that the address lines are valid and the iSBX Multimodule board should perform the specified operation.

DMA Lines (MDRQT, $\overline{\text{MDACK}}$, TDMA)

The DMA lines are the communication link between the DMA controller device on the base board and the iSBX Multimodule board. MDRQT is an active high output signal from the iSBX Multimodule board to the base board's DMA device requesting a DMA cycle. $\overline{\text{MDACK}}$ is an active low input signal to the iSBX Multimodule board from the base board DMA device acknowledging that the requested DMA cycle has been granted. TDMA is an active high output signal from the iSBX Multimodule board to the base board. TDMA is used by the iSBX Multimodule board to terminate DMA activity. The use of the DMA lines is optional as not all base boards will provide DMA channels and not all iSBX Multimodule boards will be capable of supporting a DMA channel.

Initialize Lines (Reset)

This input line to the iSBX Multimodule board is generated by the base board to put the iSBX Multimodule board into a known internal state.

Clock Lines (MCLK)

This input to the iSBX Multimodule board is a timing signal. The 10 MHz (+0%, −10%) frequency can vary from base board to base board. This clock is asynchronous from all other iSBX bus signals.

System Control Lines ($\overline{\text{MWAIT}}$, $\overline{\text{MPST}}$)

These output signals from the iSBX Multimodule board control the state of the system.

An active $\overline{\text{MWAIT}}$ (Active Low) will put the CPU on the board into wait states providing additional time for the iSBX Multimodule board to perform the requested operation. $\overline{\text{MWAIT}}$ must be generated from address (address plus chip select) information only. If $\overline{\text{MWAIT}}$ is driven active due to a glitch on the CS line during address transitions, $\overline{\text{MWAIT}}$ must be driven inactive in less than 75 ns.

The iSBX Multimodule board present ($\overline{\text{MPST}}$) is an active low signal (tied to signal ground) that informs the base board I/O decode logic that an iSBX Multimodule board has been installed.

Address and Chip Select Lines

The address and chip select lines are made up of two groups of signals.

Address Lines: MA0–MA2

Chip Select Lines: $\overline{\text{MCS0}}$–$\overline{\text{MCS1}}$

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. The base board decodes all but the lower order three addresses in generating the iSBX Multimodule board chip selects.

Address Lines (MA0–MA2)

These positive true input lines to the iSBX Multimodule boards are generally the least three significant bits of the I/O address. In conjunction with the command and chip select lines, they establish the I/O port address being accessed. In 16-bit systems, MA0–MA2 may be connected to ADR1–ADR3 of the base board address lines.

Chip Select Lines (MCS0–MCS1/)

In an 8-bit system, these input lines to the iSBX Multimodule board are the result of the base board I/O decode logic. $\overline{\text{MCS}}$ is an active low signal which conditions the I/O command signals and thus enables communication with the iSBX Multimodule boards.

6.2 The SBX82064 Design Example

The SBX82064 Multimodule board will interface an ST506 compatible drive to any host board having an SBX connector. Two restrictions on the disk drive are that there is a maximum of 2048 cylinders and/or 8 heads. The SBX connector cannot supply the power-up current requirements of the drive. The drive must be connected directly to the power supply. The SBX82064 in Appendix C does not support DMA transfers. The version in Appendix D does support DMA transfers. Since this multimodule has a 2 kbyte sector buffer, the host microprocessor must generate a BRDY by accessing an I/O port during data transfers.

The software for communicating to the SBX board is intended to be interrupt driven. Polling for data transfers is not supported. Reading the status without an interrupt is not recommended. During the times the 82064 is accessing the sector buffer, the SBX82064 will isolate itself from the host. To support polling, a hardware generated busy pattern should be driven onto the host's data bus as is shown in the Polled Interface section. The sector buffer stores up to 2 kbytes of disk data, for multiple sector transfers. The SBX board only interfaces to one drive (for space reasons), but four drives could be used with the addition of a read data multiplexor (one IC) and the drive data cables.

Microprocessor Interface

Figure 29 is a block diagram of the SBX82064's microprocessor interface. The I/O port assignments are listed in Table 1. The functional blocks of the interface are:

Sector Buffer Isolation Logic

Wait State Logic

Sector Buffer

Sector/Drive/Head Register Logic

Table 6-1. I/O Port Assignments

Port Address	Read	Write
80H	Sector Buffer	Sector Buffer
82H	Error Reg	RWC Reg
84H	Sector Count	Sector Count
86H	Sector Number	Sector Number
88H	Cylinder Low	Cylinder Low
8AH	Cylinder High	Cylinder High
8CH	SDH Reg	SDH Reg
8EH	Status Reg	Command Reg
90H	None	None
92H	None	Asserts $\overline{BCR}$
94H	None	Asserts BRDY

NOTE:
Address assignments are determined by the host board.

Sector Buffer Isolation Logic

The host will be isolated from the SBX board whenever the 82064 is accessing its sector buffer which is enabled by $\overline{BCS}$. The host's control signals, $\overline{RD}$, $\overline{WR}$, $\overline{MCSO}$, and $\overline{MCS1}$ and data bus are also disabled at the same time to prevent any data in the sector buffer from being corrupted. The host should wait for an interrupt before reading the 82064's Status register. Attempting to read the SBX board while $\overline{BCS}$ is active will return invalid data, since the SBX board will have the data bus tri-stated.

Wait State Logic

The wait state logic drives the 'not ready' line, $\overline{MWAIT}$, active whenever the host reads the SBX board. $\overline{MWAIT}$ does not go active for buffer or 82064 register writes. This logic was required for two reasons. First, a delayed read is generated, because the address setup to $\overline{RD}$ margin of the SBX bus is less than the 82064's needs (50 ns vs 100 ns). Second, the $\overline{RD}$ to data valid access period of the 82064 (375 ns), is greater than the SBX bus' full speed read cycle (275 ns) permits.

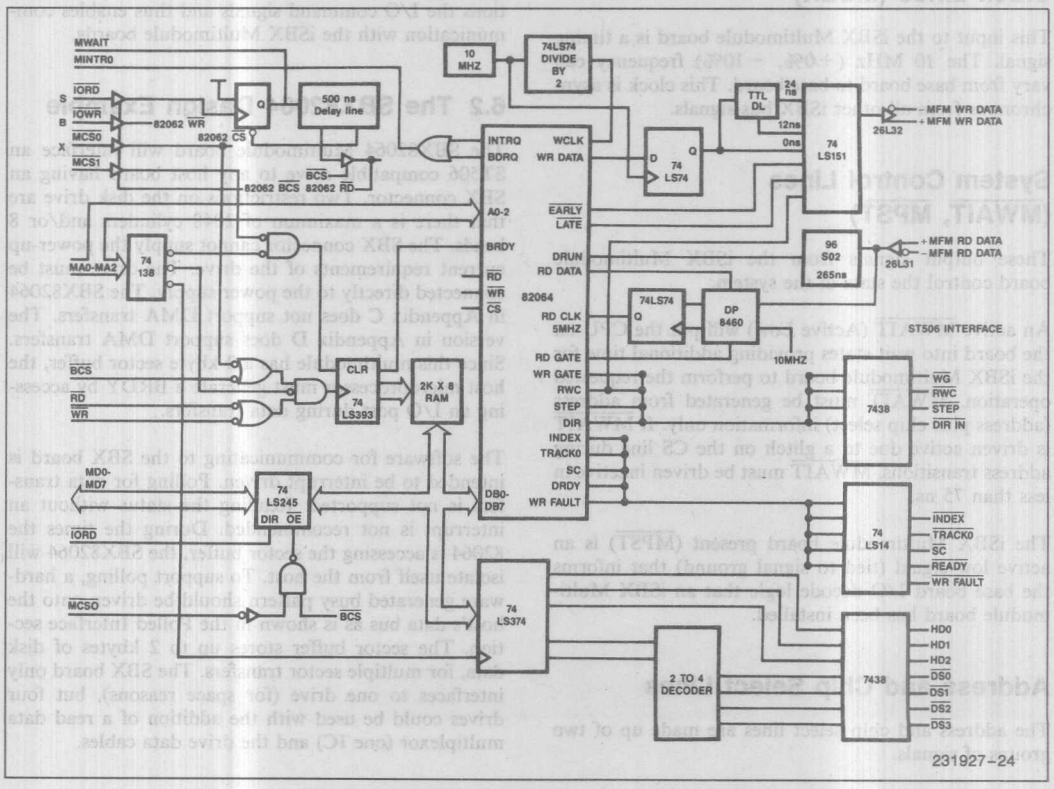


Figure 29. 82064 SBX Multimodule Diagram

$\overline{MWAIT}$ is deactivated after allowing for the delayed $\overline{RD}$ and the access period of the 82064. This delay is accomplished with a 500 ns delay line. The first tap at 100 ns generates the read request to allow for the address setup margin. The next tap 400 ns later removes $\overline{MWAIT}$ to allow the host to continue.

Sector Buffer

The sector buffer consists of an address counter (using '1s393's) and a 2 kbyte static RAM. The address counter is incremented on the trailing edge of a valid $\overline{RD}$ or $\overline{WR}$ cycle, either host microprocessor or 82064 initiated. The counter is reset by a hardware reset, the 82064 buffer reset $\overline{BCR}$, or by accessing an I/O port to provide software control. The 82064 will issue $\overline{BCR}$ each time $\overline{BCS}$ changes state (i.e. twice per sector). Resetting the buffer counter can be put under software control for multiple sector transfers. BRDY going high tells the 82064 that the buffer is available for its use. BRDY is generated by the address counter, by filling or emptying the entire buffer in multiple sector transfers, or from an I/O port when single sector transfers are done (since single sectors won't use all 2 kbytes of the buffer, the hardware signal will not be generated). When the 82064 is using the buffer, $\overline{BCS}$ will be low, and the $\overline{RD}$ or $\overline{WR}$ line will be pulsed every 1.6 microseconds.

When the 82064 is using the buffer it prevents access by the host by tristating the read, write, select and data lines with a low on $\overline{BCS}$.

SDH Register Logic

The drive and head select bits must be latched externally to the 82064, since these outputs are not provided. An 8 bit latch is strobed on the trailing edge of the $\overline{WR}$ pulse when the SDH register is selected. The two drive select bits are then demultiplexed to provide a one of four drive select line. If multiple drives are used then these outputs would also be used to select which disk's read data line would be gated into the PLL.

Interrupts

While the interrupt line is programmable (to notify of an end of command or data transfer request for the Read Sector command only), software will ensure that the interrupt from the 82064 signifies command termination. The BDRQ line is OR'ed with the 82064's INTRQ line or BDRQ can generate its own interrupt. BDRQ is also gated off-board for a DMA controller.

Disk Interface

Figure 30 is a block diagram of the interface between the 82064 and the disk drive. The functional blocks are:

Write Data Logic

Read Data Logic (PLL)

Drive Control

Write Data Logic

The WR DATA output requires a D flip-flop clocked at 10 MHz to complete the conversion of data to MFM. The output of this D flip-flop is true MFM and is sent to a delay line. A delay line determines the amount of delay for precompensation. No delay corresponds to shifting the data bit early; the first tap is approximately 12 ns of delay and is the "normal", or no delay and the second tap provides 12 ns of delay, referenced to the "normal" write data. Which output is selected is determined by the states on RWC, $\overline{Early}$ and $\overline{Late}$. This function was generated with a 74s151 multiplexer. When RWC is inactive $\overline{EARLY}$ and $\overline{LATE}$ only select "normal" data since they are always active. The precompensated write data is then driven onto the data cable by an RS-422 driver.

Read Data Logic

The PLL generates the RD CLOCK that is used to decode the serial MFM data from the drive. A selected drive issues read data, unless WR GATE is active. A one-shot generates a pulse of 220–270 ns to provide the DRUN input. Only during an all zero's or one's field will the DRUN input stay high, as it will be retriggered every 200 ns (the minimum distance that separates continuous clock and data bits). As soon as DRUN is determined to be valid, the RD GATE output will go active, switching the PLL from the 10 MHz local clock input to disk data. The PLL will synchronize to the incoming serial data and generate a Read Clock of the proper timing and phase. The 82064 will then start to search for the address mark which is indicated by DRUN going low at the address mark.

No detail is provided herein on PLL design, as it is beyond the scope of this document. PLL design should be left to experienced designers, since minute changes in temperature and component values will drastically affect the soft error rate. As an alternative, several companies manufacture very high speed PLL chips for MFM encoded disk drives. Besides being fairly easy to design in, they reduce the number of components and board area needed for the sophisticated PLL.

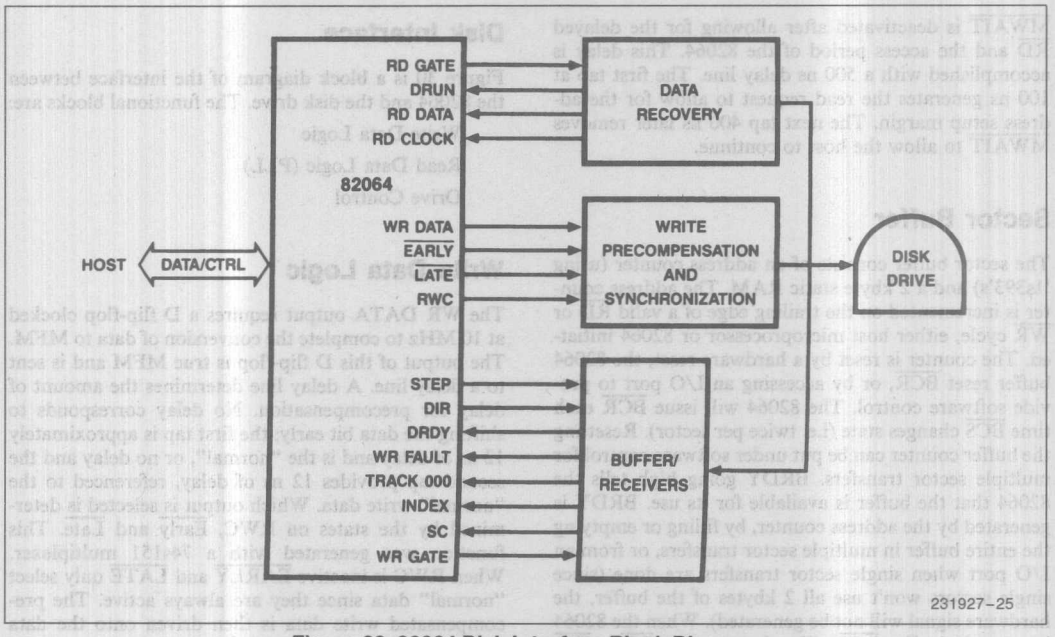


Figure 30. 82064 Disk Interface Block Diagram

6.3 Software Driver Overview

Presented in Appendix B is a listing of the software used to exercise the SBX 82064 board. Communication between the host software and the SBX driver routine is done through a structure located in system RAM. The host routine fills in required parameters, then passes the address of this communication block to the SBX driver routine. The driver routine pulls necessary values from this command block (CBL), executes a disk operation, then fills the CBL with the 82064's register contents, plus status and error information. The command block structure is shown in Figure 31.

Command	Byte
Rwc Reg	Byte
Sector Cnt.	Byte
Sector Num.	Byte
Cyl Low	Byte
Cyl High	Byte
SDH Reg	Byte
Status Reg	Byte
Error Reg	Byte
Host Buffer	Pointer

Figure 31

The host board did not have a DMA controller available, so an interrupt is issued from the BDRQ line and OR'ed with the 82064's interrupt line as interrupt sources were limited by the host. When an interrupt occurs, the interrupt procedure checks for either a data transfer, and executes it, or the completion of the command. If the interrupt signifies command completion, the interrupt procedure fills the command block with the 82064's task, status and error registers.

In this example, the host software examines one byte in the command block and until this byte is changed to a 00, no other command blocks will be passed to the disk driver routine. An alternative would be to issue a software interrupt to notify the microprocessor that the disk operation has finished and the command block contains parameters from the last operation and that a new disk command could start.

The driver for this example allows polling for non-data transfer commands, and must use interrupts for data transfers. As mentioned earlier, microprocessor intervention is required since the sector buffer is much larger than one sector and will not generate a BRDY. The microprocessor must write to an I/O port, which sets BRDY, after each host to sector buffer transfer. An

actual software implementation would not include the polling and interrupt routines together, as only one method would generally be used.

The calling routine, which would normally be a directory program, places the values for which sector, number of sectors, etc., in the CBL. The disk routine is called and the address of this structure is passed on the stack. The disk driver places these parameters in the 82064's Task registers and initiates a command.

If the interrupt driven method was chosen, the disk driver routine returns to the calling routine. This permits other processing to be performed while the disk is executing a command. At some point, an interrupt will be generated, either from BRDY or INTRQ. Control will pass to the driver and the status register will be checked. If a data transfer is needed, either the microprocessor can transfer data or a DMA controller can perform the function. Once the transfer of data to the buffer is finished the microprocessor must set BRDY through an I/O port.

APPENDIX A
ST506 INTERFACE

THE ST506 INTERFACE

The ST506 interface is a modified version of Shugarts floppy disk drive interface and has been promoted by Seagate Technology. This interface is intended to be easy and low in cost to implement, yet provide a medium level of performance. The interface rigidly defines several areas: the hardware interconnects, the data transfer rate, the data encoding method, and how the disk is formatted.

Data Transfer Rate

The data transfer rate depends upon the linear bit density of the disk media and the speed at which the disk spins. ST506 specifies a 5 Mbit/second transfer rate. The typical ST506 drive has a nominal linear density of 10,416 bytes and a disk speed of 3600 rpm, which yields a 5 Mbit/second data transfer rate. No deviation from 5 M/bits second is allowed.

Increasing the linear density to increase storage capacity would require a decrease in disk speed. Otherwise, the data rate would increase. This decrease in disk speed would cause access times to increase, which many would deem unacceptable. To increase storage capacity, and remain ST506 compatible, either the number of cylinders and/or the number of platters can increase.

Data Encoding

ST506 requires that the serial data, sent between the drive and the controller, be encoded according to MFM rules. The basic unit of storage is a bit cell, which stores one bit information. This bit cell is divided into two halves, consisting of a clock bit and a data bit (see Figure A-1).

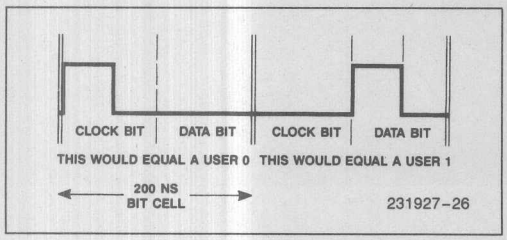


Figure A-1

The encoding rules for MFM are fairly simple:

1. A clock bit is written when the previous and the current bit cell does not contain a data bit.
2. A data bit is written whenever there is a "one" from the user.

Sync fields are composed of zeroes which generates a series of clock bits in the bit cell's. A phase lock loop locks on to the data stream during this period and generates a signal of the proper phase and frequency which is used to decode the combined clock and data serial data stream.

Disk Format

All disk media must be written with a specified format so that data may be reliably stored and retrieved. The smallest unit of controller accessible data is the sector which typically contains sync fields, ID fields, and a data field, and buffer fields.

The format of the disk required by ST506 is shown in Figure A-2. It should be noted that this format is fixed in the 82064. The user has options only for GAP1 and 3 length (when changing sector size or ECC) and whether to have 82064 CRC checking or user supplied ECC syndrome bits.

Gap 1 – Index Gap

Gap 1 serves two purposes. The first is to allow for variations in the index pulse timing due to motor speed variations. The second purpose is to allow a small delay to permit a different head to be selected without missing a sector. This is more of a data transfer optimization function and requires the disk controller to know which head is to be selected, when the last sector of a track has been read, and the next logical sector in the file exists on another platter. The 82064 does not switch heads automatically. Whether this scheme can be used or not depends upon the μP being able to alter one register in the 82064, before the next sector passes beneath the heads.

This gap is typically 12 bytes long and is written by the 82064 as 4E Hex.

Gap 2 – Write Splice Gap

This gap follows the CRC bytes of the ID field and continues up to the data field address mark. When updating a previously written sector, motor speed variations could turn on the write coil, as the head was passing over the ID field. This gap prevents this from occurring. The value written is OOH and also serves as the PLL sync field for the data field. The minimum value is determined by the "lock up" performance of the PLL. The 82064 writes sixteen bytes for this field once WG is activated. The user has no control over this field.

Gap 3 – Post Data Field Gap

Gap 3 is very similar to Gap 2 as it is used as a speed tolerance buffer also. Without this gap, and with the motor speed varying slightly, it would be possible for the upcoming sector's sync field and ID field to be overwritten. This value is '4E' H and is typically 15 bytes long. The 82064's Gap 3 length is programmable. The exact value is dependent upon several factors. Refer to 82064 Format command, Software Section: General Programming Section.

Gap 4 – Track Buffer Gap

This gap follows the last sector on a track and is written until an index pulse is received. Its purpose is to prevent the last sector from overflowing past the index gap, and absorb track length variations when ECC is used (ECC uses more bytes than CRC). The value is '4E' H and is about 320 bytes when CRC and 256 byte sectors are used. The 82064 writes this field only during formatting. The user has no control over the number of bytes written with the 82064.

ID Fields

The controller uses ID fields to locate any individual sector. An address mark of two bytes precedes the ID field and the data field in a sector. An address mark tells the controller the nature of the upcoming information. ID fields are used by the disk controller and are not passed to the host.

Sector Interleaving

Sector interleaving occurs when logical sectors are in a non-sequential order, which is determined during formatting. An advantage is that there is a delay between logically sequential sectors. This delay can be used for data processing and then deciding if the next sector should be read. Without interleaving, the next sector could slip by, imposing a one revolution delay (approx. 16.7 ms). An additional benefit to this delay is that bus utilization is reduced by spreading the data transfer over a greater amount of time. The delay between sectors can be determined as follows:

$$\frac{\text{1 Revolution Period}}{\text{Sectors/Track}} \times (\text{Interleave factor} - 1) = \text{Delay}$$

For the typical ST506 drive with four-way interleaving this yields 1.57 ms of delay.

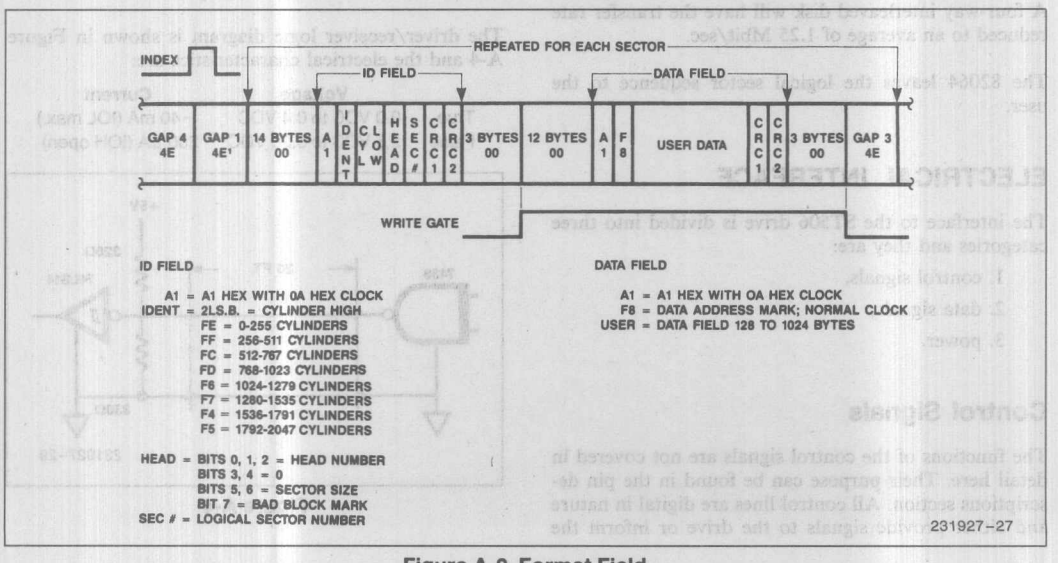


Figure A-2. Format Field

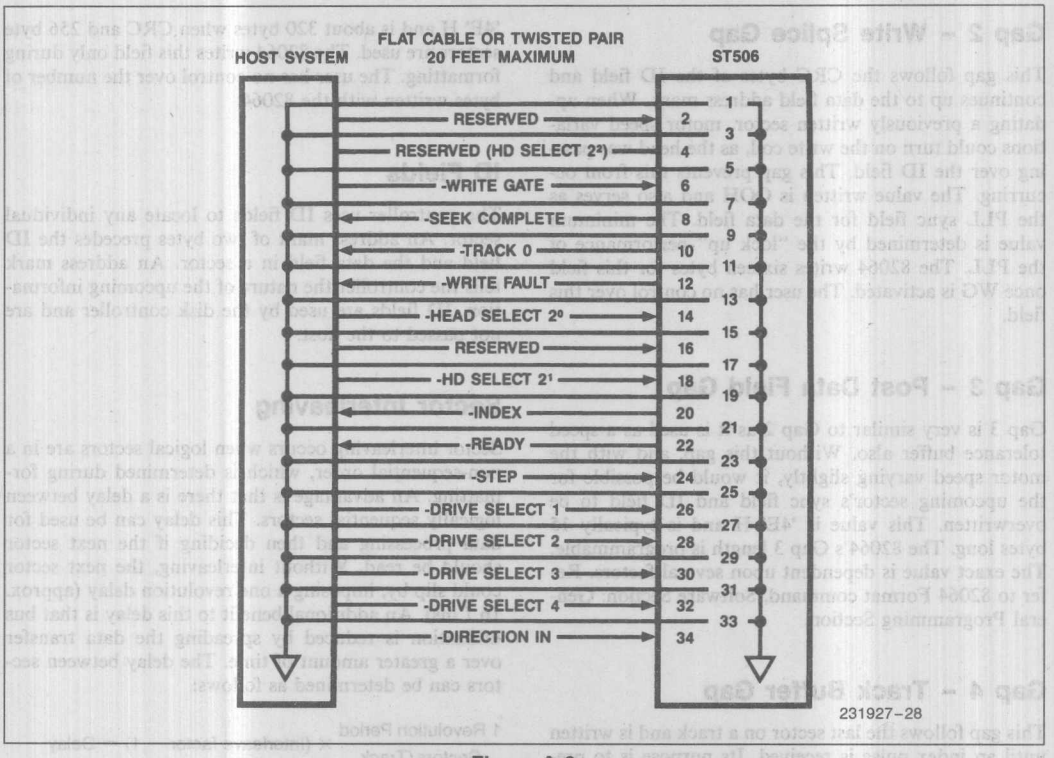


Figure A-3

The disadvantage to interleaving is that file transfers take longer, which may slow down the overall system. A four-way interleaved disk will have the transfer rate reduced to an average of 1.25 Mbit/sec.

The 82064 leaves the logical sector sequence to the user.

ELECTRICAL INTERFACE

The interface to the ST506 drive is divided into three categories and they are:

1. control signals,
2. data signals,
3. power.

Control Signals

The functions of the control signals are not covered in detail here. Their purpose can be found in the pin descriptions section. All control lines are digital in nature and either provide signals to the drive or inform the host of certain conditions. A diagram of the 34 pin control connector is shown in Figure A-3.

The driver/receiver logic diagram is shown in Figure A-4 and the electrical characteristics are:

	Voltage	Current
True	0.0 VDC to 0.4 VDC	−40 mA (IOL max.)
False	2,5 VDC to 5.25 VDC	250 μA (IOH open)

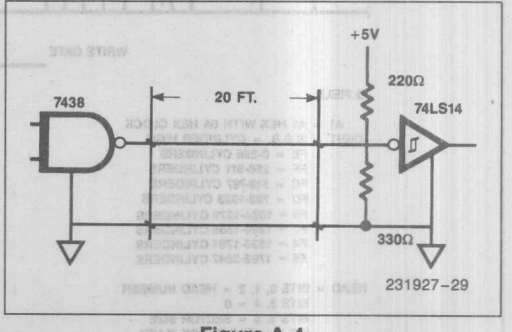


Figure A-4

Data Signals

The lines associated with the transfer of read/write data between the host and the drive are differential in nature and may not be multiplexed between drives. There is one pair of balanced lines for each read and write data line per drive and must conform to the RS-422 specification. Figure A-5 shows the receiver/transmitter combination.

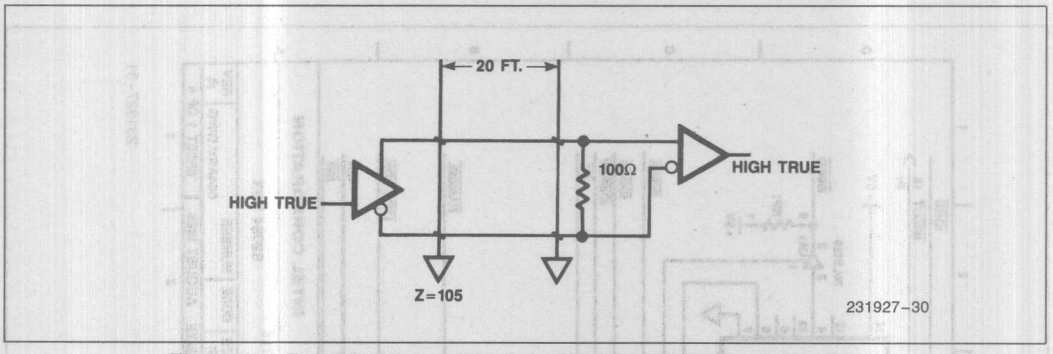
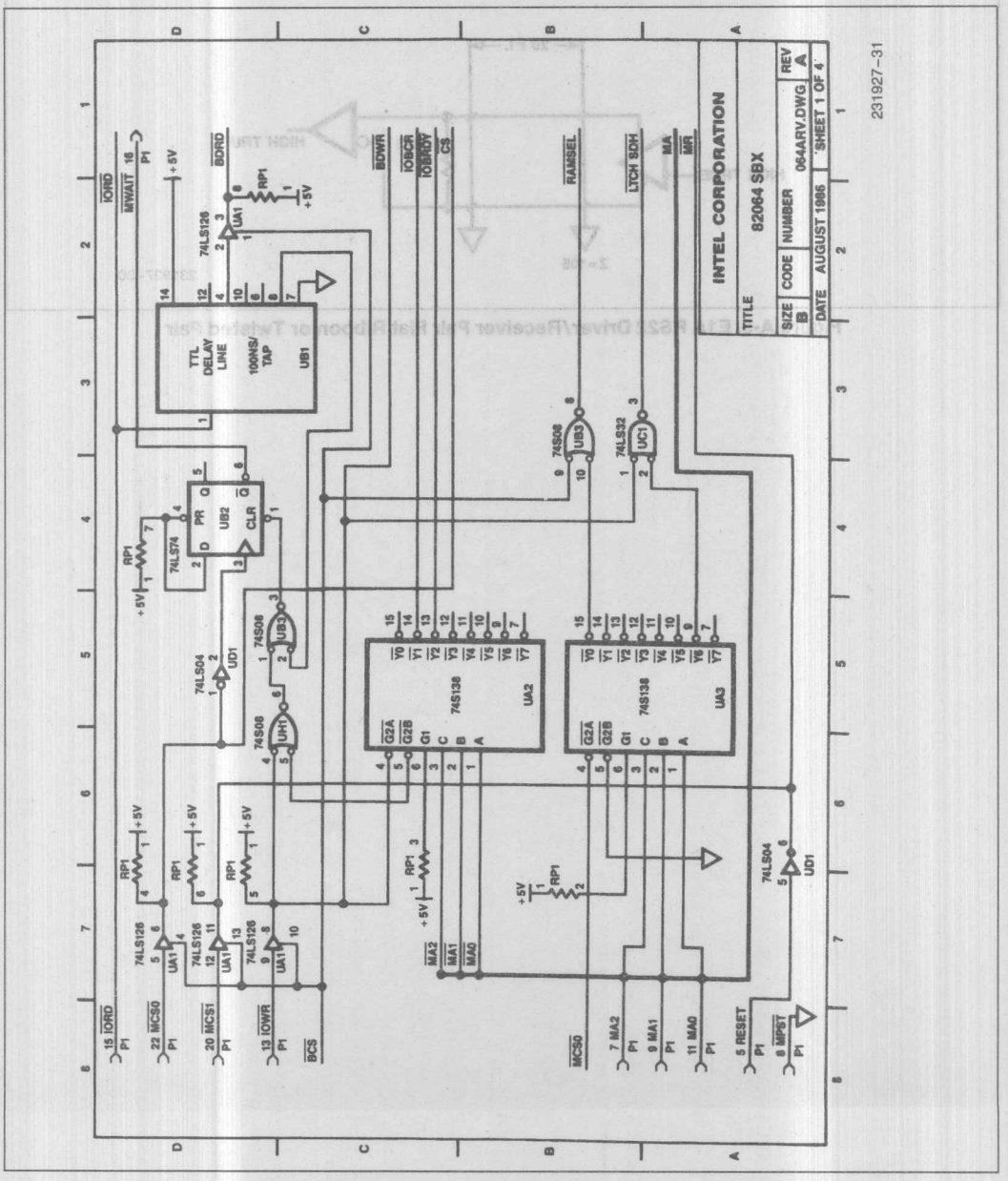
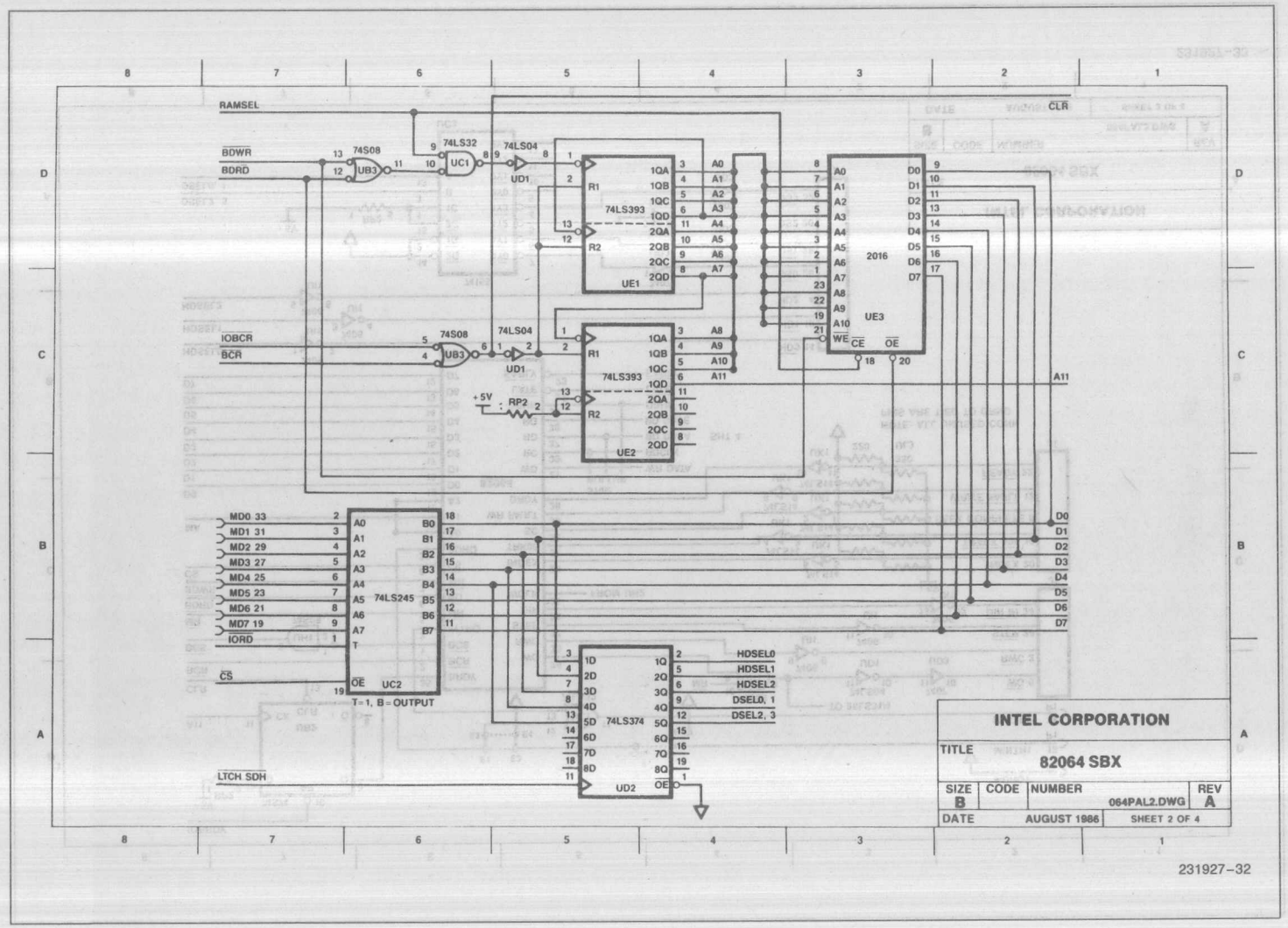


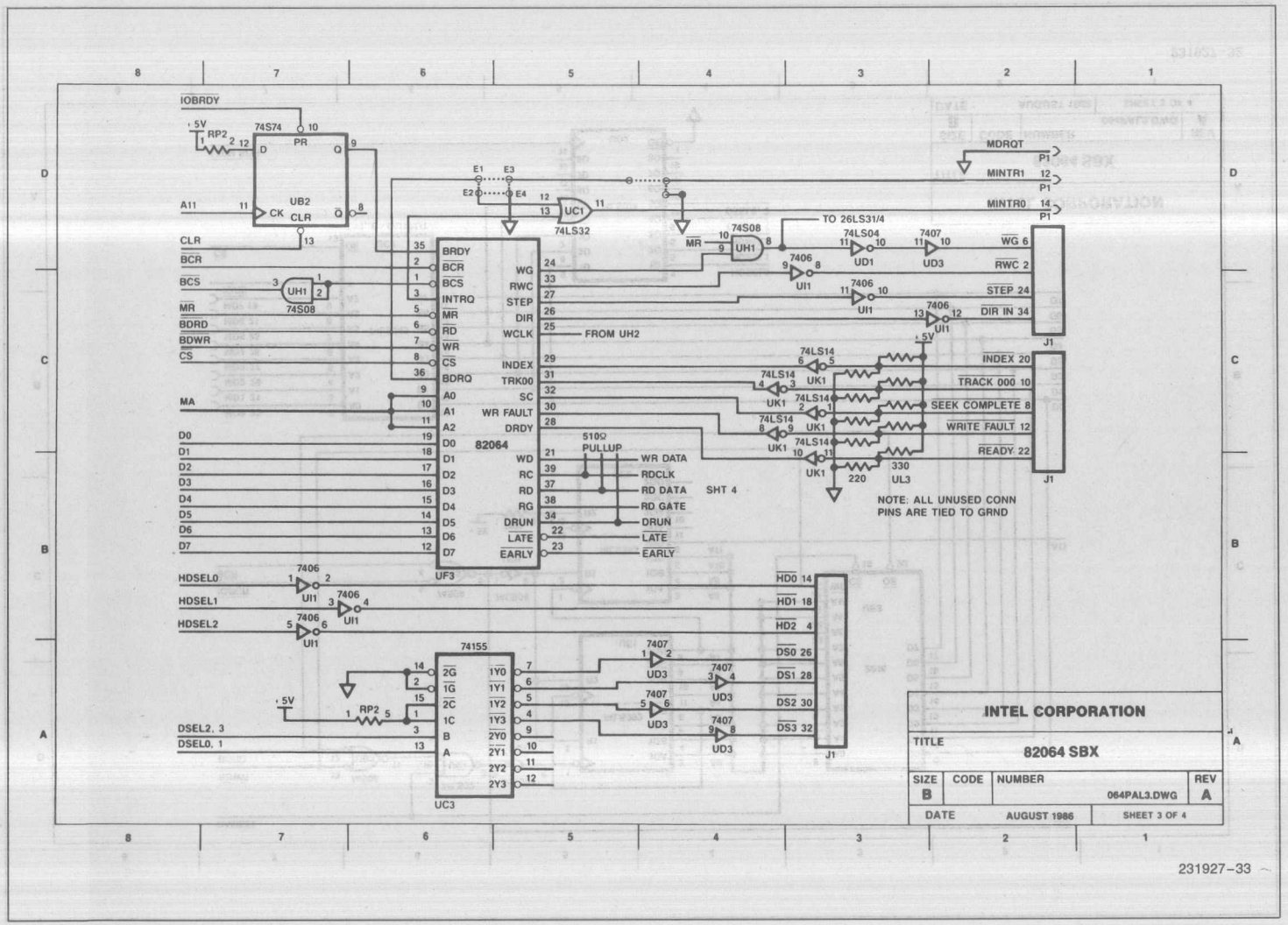
Figure A-5. E1A RS22 Driver/Receiver Pair Flat Ribbon or Twisted Pair

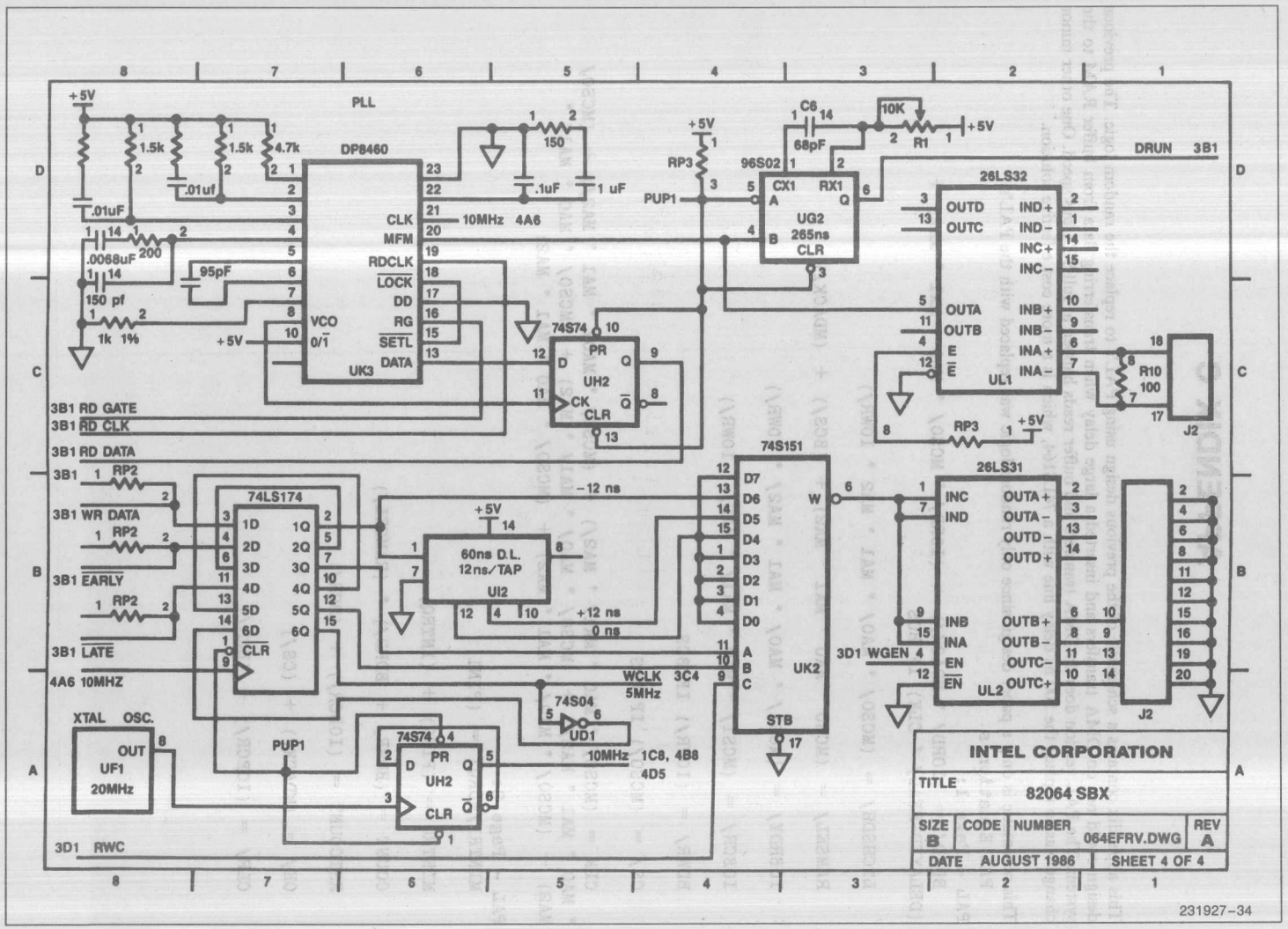
APPENDIX B
SCHEMATICS




RAMSEL
BDWR
BDRD
74S08
UB3
74LS32
UC1
74LS04
UD1
74LS393
UE1
UE2
2016
UE3
CLR
IOBCR
BCR
+5V
RP2
A11
MD0 33
MD1 31
MD2 29
MD3 27
MD4 25
MD5 23
MD6 21
MD7 19
IORD
CS
74LS245
UC2
T=1, B=OUTPUT
74LS374
UD2
HDSEL0
HDSEL1
HDSEL2
DSEL0, 1
DSEL2, 3
LTCH SDH
D0
D1
D2
D3
D4
D5
D6
D7
INTEL CORPORATION
TITLE
82064 SBX
SIZE B
CODE
NUMBER 064PAL2.DWG
REV A
DATE AUGUST 1986
SHEET 2 OF 4

231927-32


IOBRDY
+5V
RP2
74S74
UB2
A11
CLR
BCR
BCS
MR
BDRD
BDWR
CS
MA
D0
D1
D2
D3
D4
D5
D6
D7
BRDY
INTRQ
BDRQ
A0
A1
A2
82064
WG
RWC
STEP
DIR
WCLK
INDEX
TRK00
SC
WR FAULT
DRDY
WD
RC
RD
RG
DRUN
LATE
EARLY
FROM UH2
510Ω PULLUP
WR DATA
RDCLK
RD DATA
RD GATE
DRUN
LATE
EARLY
SHT 4
UF3
UC1
74LS32
UH1
74S08
MDRQT
MINTR1
MINTR0
TO 26LS31/4
74LS04
UD1
UD3
7407
7406
UI1
WG 6
RWC 2
STEP 24
DIR IN 34
J1
74LS14
UK1
INDEX 20
TRACK 000 10
SEEK COMPLETE 8
WRITE FAULT 12
READY 22
220
330
UL3
NOTE: ALL UNUSED CONN PINS ARE TIED TO GRND
HDSEL0
HDSEL1
HDSEL2
HD0 14
HD1 18
HD2 4
DS0 26
DS1 28
DS2 30
DS3 32
74155
UC3
DSEL2, 3
DSEL0, 1
INTEL CORPORATION
TITLE
82064 SBX
SIZE B
CODE
NUMBER
064PAL3.DWG
REV A
DATE AUGUST 1986
SHEET 3 OF 4
231927-33


PLL
DP8460
UK3
VCO
CLK
MFM
RDCLK
LOCK
DD
RG
SETL
DATA
10MHz 4A6
74S74
UH2
PR
CK
CLR
3B1 RD GATE
3B1 RD CLK
3B1 RD DATA
3B1 WR DATA
3B1 EARLY
3B1 LATE
4A6 10MHZ
74LS174
60ns D.L.
12ns/TAP
UI2
+12 ns
0 ns
-12 ns
74S151
UK2
STB
WCLK 3C4
5MHz
74S04
UD1
10MHz 1C8, 4B8
4D5
XTAL OSC.
UF1
20MHz
PUP1
3D1 RWC
96S02
UG2
265ns
CX1
RX1
68pF
10K
R1
DRUN 3B1
26LS32
UL1
26LS31
UL2
3D1 WGEN
R10
100
J2
INTEL CORPORATION
TITLE
82064 SBX
SIZE B
CODE
NUMBER 064EFRV.DWG
REV A
DATE AUGUST 1986
SHEET 4 OF 4

231927-34

APPENDIX C

This appendix contains a schematic of the previous design using PAL's to replace the random logic. The previous design could not do DMA transfers and inserted a large delay when transferring data from buffer RAM to the system. The PAL version does do DMA transfers and buffer reads happen at full SBX bus speed. One other minor change was to replace the 500 ns delay line with a 74LS164, which is a more cost effective solution.

This schematic is only a paper design since only random logic was replaced with the PAL's.

```
   PAL Equation's
PAL - Page 1:
   BDRD/ = (IORD/ * MDACK/) + (IORD/ * MCS0/ * MA0 * MA1 * MA2) +
(DELAYED-READ/ * CLK) IF BCS

   LTCHSDH/ = (MCS0/ * MA0/ * MA1 * MA2 * IOWR/)

   RAMSEL/ = (MCS0 * MA0 * MA1 * MA2) + (BCS/) + (MDACK/)

   IOBRDY/ = (MCS1/ * MA0/ * MA1 * MA2/ * IOWR/)

   IOBCR/ = (MCS1/ * MA0 * MA1/ * MA2/ * IOWR/)

   BDWR/ = (IOWR/) IF BCS

   CS/ = (MCS0/) IF BCS

   CLK = (MCS0/ * MA0 * MA1/ * MA2/) + (MCS0/ * MA0/ * MA1 * MA2/) + (MCS0/
* MA0 * MA1 * MA2/) + (MCS0/ * MA0/ * MA1/ * MA2) + (MCS0/ * MA0 * MA1/ *
MA2) + (MCS0/ * MA0/ * MA1 * MA2) + (MCS0/ * MA0 * MA1 * MA2)

PAL - Page 2:
   MINTR1/MDRQT = (PIN1)

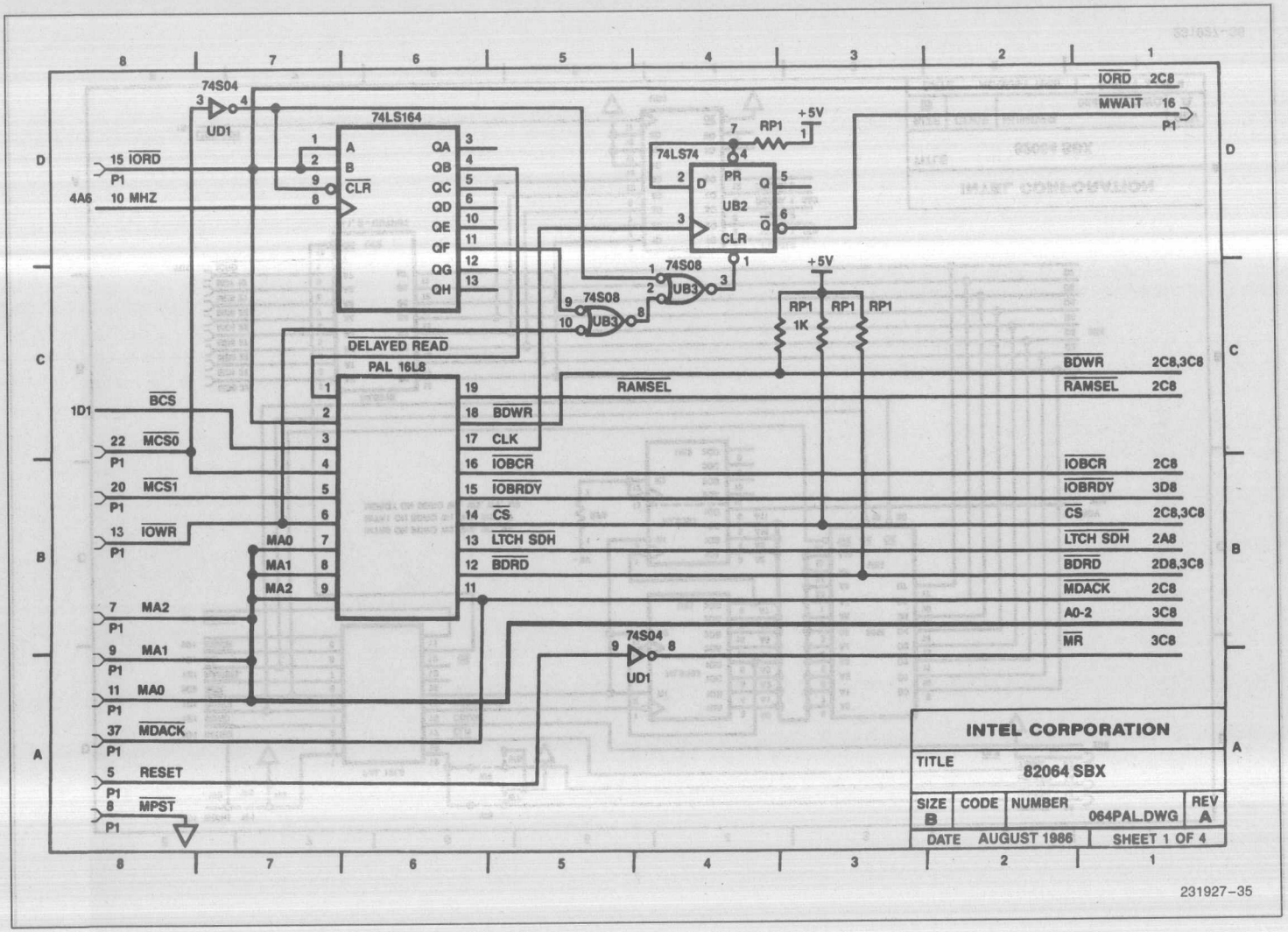
   MINTR0 = (PIN2) + (INTRQ)

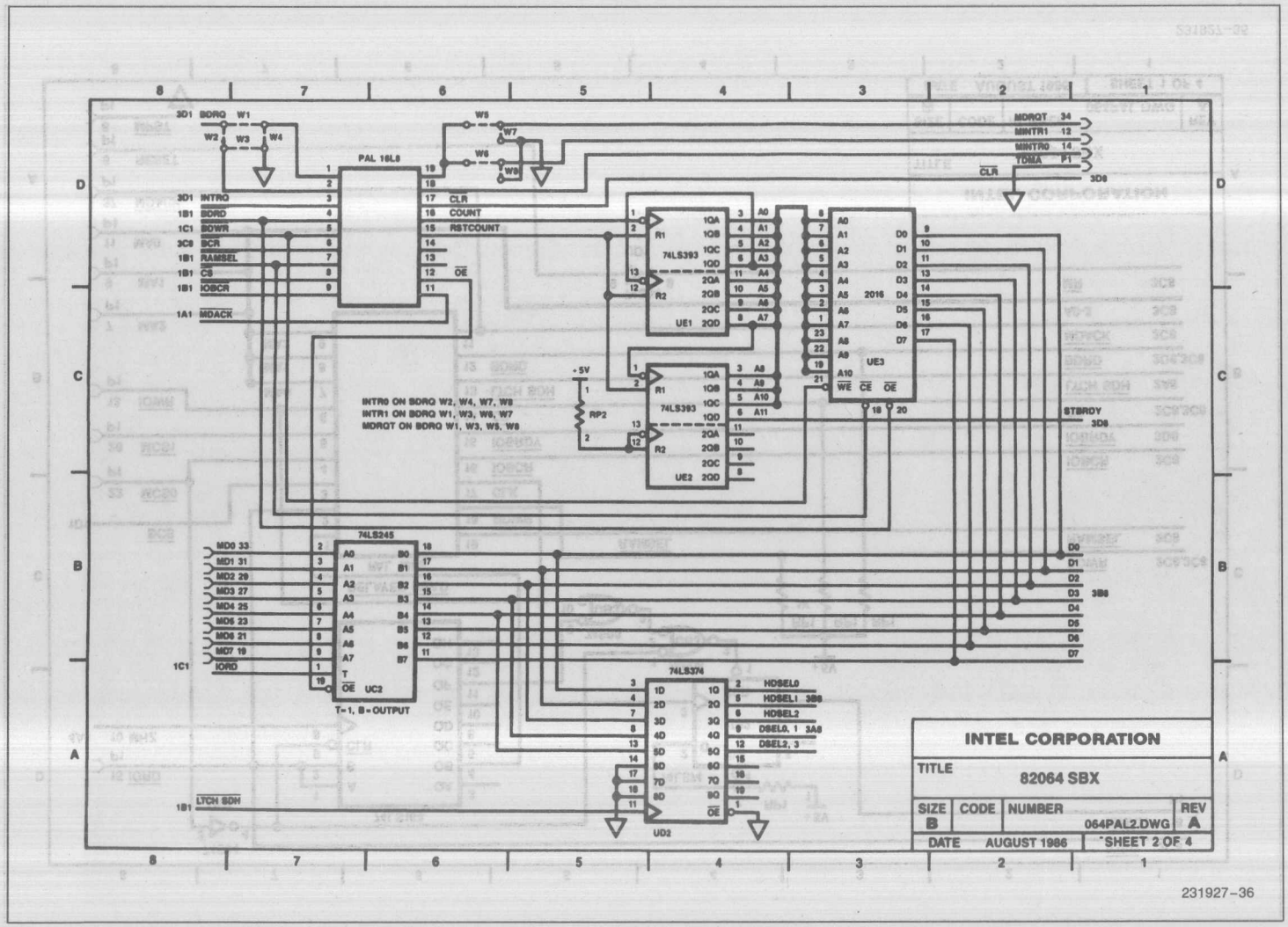
   COUNT = (BDWR/ + BDRD/) * (RAMSEL/)

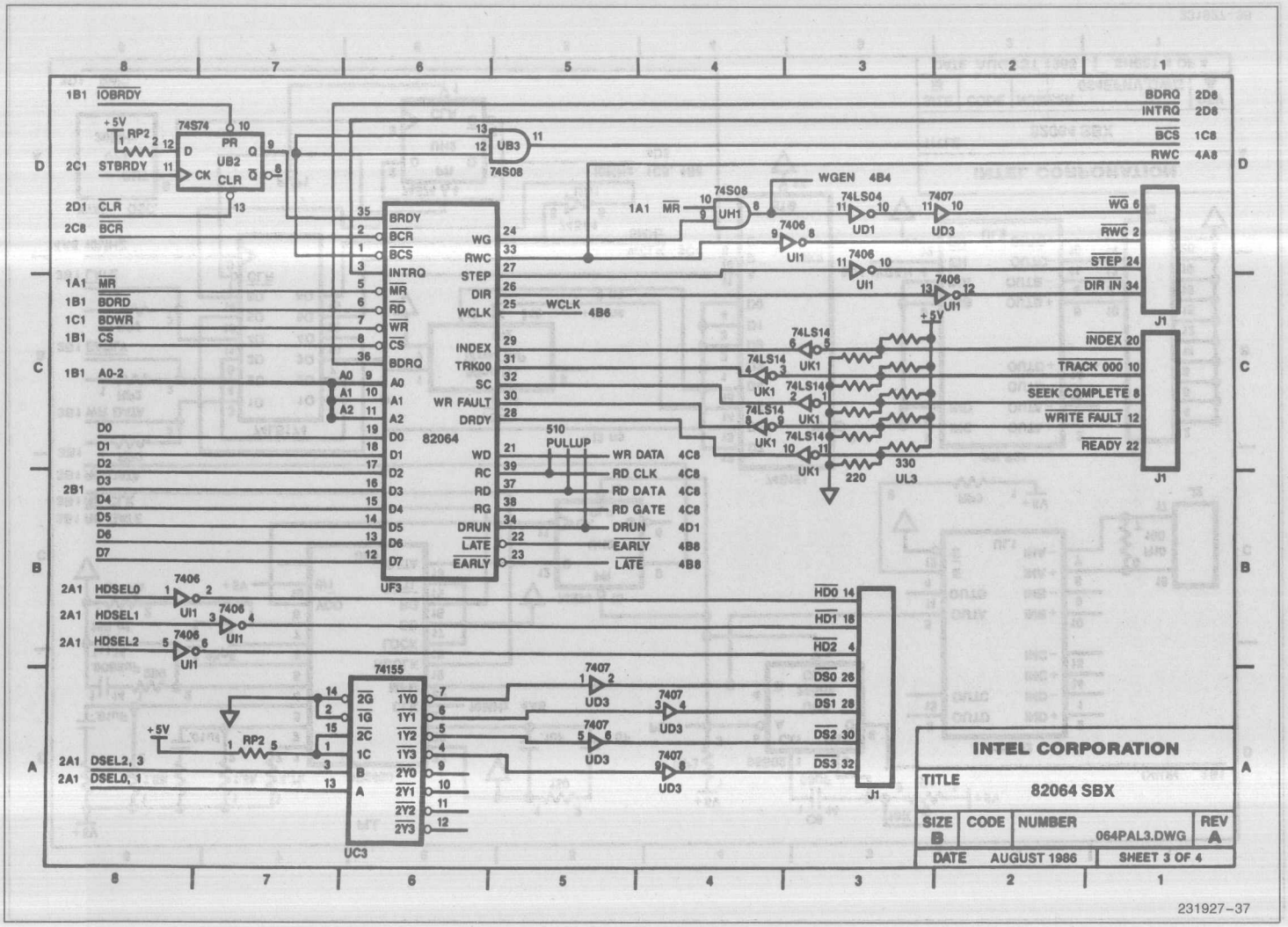
   RSTCOUNT = (IOBCR/) + (BCR/)

   OE/ = (MDACK/) + (CS/)

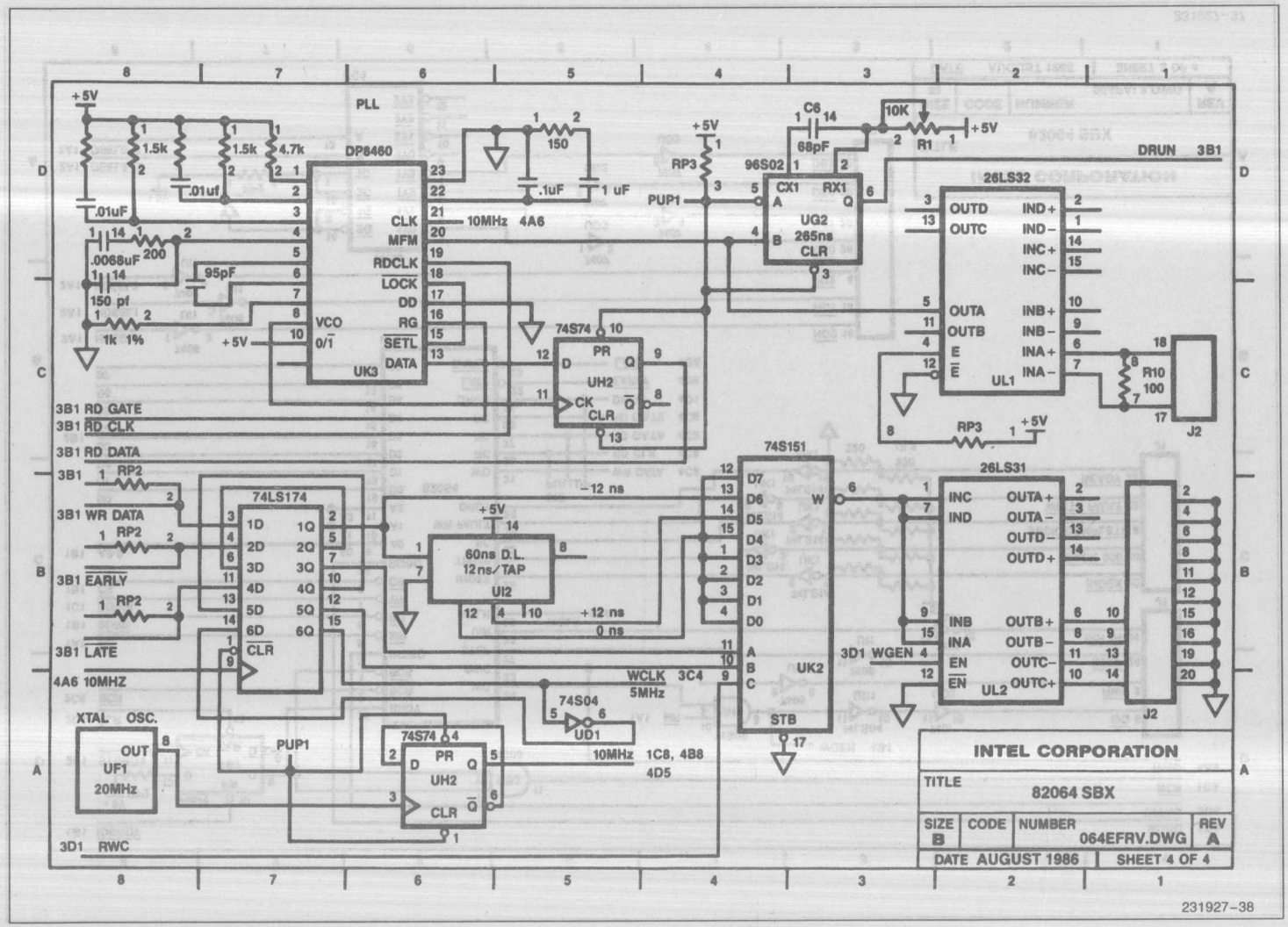
   CLR/ = (IOBCR/) + (BCR/)
```


74S04
UD1
74LS164
A
B
CLR
QA
QB
QC
QD
QE
QF
QG
QH
15 IORD
P1
4A6 10 MHZ
74LS74
PR
D
Q
UB2
CLR
RP1
+5V
74S08
UB3
RP1 1K
DELAYED READ
PAL 16L8
1D1 BCS
22 MCS0
20 MCS1
13 IOWR
MA0
MA1
MA2
7 MA2
9 MA1
11 MA0
37 MDACK
5 RESET
8 MPST
RAMSEL
BDWR
CLK
IOBCR
IOBRDY
CS
LTCH SDH
BDRD
IORD 2C8
MWAIT 16
BDWR 2C8,3C8
RAMSEL 2C8
IOBCR 2C8
IOBRDY 3D8
CS 2C8,3C8
LTCH SDH 2A8
BDRD 2D8,3C8
MDACK 2C8
A0-2 3C8
MR 3C8
INTEL CORPORATION
TITLE
82064 SBX
SIZE B
CODE
NUMBER
064PAL.DWG
REV A
DATE AUGUST 1986
SHEET 1 OF 4
231927-35


PAL 16L8
74LS393
74LS245
74LS374
2016
INTR0 ON BDRQ W2, W4, W7, W8
INTR1 ON BDRQ W1, W3, W6, W7
MDRQT ON BDRQ W1, W3, W5, W8
T=1, B=OUTPUT
INTEL CORPORATION
TITLE
82064 SBX
SIZE B
CODE
NUMBER 064PAL2.DWG
REV A
DATE AUGUST 1986
SHEET 2 OF 4
231927-36


1B1 IOBRDY
+5V
RP2
74S74
2C1 STBRDY
2D1 CLR
2C8 BCR
UB2
UB3
74S08
BDRQ 2D8
INTRQ 2D8
BCS 1C8
RWC 4A8
WGEN 4B4
1A1 MR
UH1
74LS04
UD1
7407
UD3
7406
UI1
WG 6
RWC 2
STEP 24
DIR IN 34
WCLK 4B6
J1
INDEX 20
TRACK 000 10
SEEK COMPLETE 8
WRITE FAULT 12
READY 22
74LS14
UK1
330
220
UL3
1B1 BDRD
1C1 BDWR
1B1 CS
1B1 A0-2
2B1
BRDY
BCR
BCS
INTRQ
MR
RD
WR
CS
BDRQ
A0
A1
A2
D0
D1
D2
D3
D4
D5
D6
D7
82064
WG
RWC
STEP
DIR
WCLK
INDEX
TRK00
SC
WR FAULT
DRDY
WD
RC
RD
RG
DRUN
LATE
EARLY
UF3
510
PULLUP
WR DATA 4C8
RD CLK 4C8
RD DATA 4C8
RD GATE 4C8
DRUN 4D1
EARLY 4B8
LATE 4B8
2A1 HDSEL0
2A1 HDSEL1
2A1 HDSEL2
HD0 14
HD1 18
HD2 4
DS0 26
DS1 28
DS2 30
DS3 32
74155
2G
1G
2C
1C
B
A
1Y0
1Y1
1Y2
1Y3
2Y0
2Y1
2Y2
2Y3
UC3
2A1 DSEL2, 3
2A1 DSEL0, 1
INTEL CORPORATION
TITLE
82064 SBX
SIZE B
CODE
NUMBER
064PAL3.DWG
REV A
DATE AUGUST 1986
SHEET 3 OF 4

231927-37


PLL
DP8460
UK3
VCO
0/1
CLK
MFM
RDCLK
LOCK
DD
RG
SETL
DATA
10MHz 4A6
+5V
1.5k
4.7k
.01uF
.0068uF
200
95pF
150 pf
1k 1%
150
.1uF
1 uF
RP3
PUP1
96S02
CX1
RX1
UG2
265ns
CLR
C6
68pF
10K
R1
DRUN 3B1
26LS32
OUTD
OUTC
IND+
IND−
INC+
INC−
OUTA
OUTB
INB+
INB−
INA+
INA−
UL1
R10
100
J2
74S74
UH2
PR
CK
CLR
3B1 RD GATE
3B1 RD CLK
3B1 RD DATA
3B1 WR DATA
3B1 EARLY
3B1 LATE
RP2
74LS174
4A6 10MHZ
60ns D.L.
12ns/TAP
UI2
−12 ns
+12 ns
0 ns
74S151
D7
D6
D5
D4
D3
D2
D1
D0
W
UK2
STB
26LS31
INC
IND
INB
INA
EN
OUTA+
OUTA−
OUTD−
OUTD+
OUTB+
OUTB−
OUTC−
OUTC+
UL2
3D1 WGEN
WCLK 3C4
5MHz
74S04
UD1
10MHz 1C8, 4B8
4D5
XTAL OSC.
OUT
UF1
20MHz
3D1 RWC
INTEL CORPORATION
TITLE
82064 SBX
SIZE
B
CODE
NUMBER
064EFRV.DWG
REV
A
DATE AUGUST 1986
SHEET 4 OF 4
231927-38

Universal Peripheral Interface Slave Microcontrollers

5

5 Universal Peripheral Interface Slave Microcontrollers

UPI™-452
CHMOS PROGRAMMABLE I/O PROCESSOR

83C452 - 8K × 8 Mask Programmable Internal ROM

87C452P - 8K × 8 Piggyback EPROM

80C452 - External ROM/EPROM

- 83C452/87C452P/80C452:3.5 to 16 MHz Clock Rate
- Software Compatible with the MCS-51 Family
- 128-Byte Bi-Directional FIFO Slave Interface
- Two DMA Channels
- 256 × 8-Bit Internal RAM
- 34 Additional Special Function Registers
- 40 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Boolean Processor
- Bit Addressable RAM
- 8 Interrupt Sources
- Programmable Full Duplex Serial Channel
- 64K Program Memory Space
- 64K Data Memory Space
- 68-Pin PGA
  (See Packaging Spec., Order: #231369)

The Intel UPI-452 (Universal Peripheral Interface) is a 68 pin CHMOS Slave I/O Processor with a sophisticated bi-directional FIFO buffer interface on the slave bus and a two channel DMA processor on-chip. The UPI-452 is the newest member of Intel's UPI family of products. It is a general-purpose slave I/O Processor that allows the designer to grow a customized interface solution.

The UPI-452 contains a complete 80C51 with twice the on-chip data and program memory. The sophisticated slave FIFO module acts as a buffer between the UPI-452 internal CPU and the external host CPU. To both the external host and the internal CPU, the FIFO module looks like a bi-directional bottomless buffer that can both read and write data. The FIFO manages the transfer of data independent of the UPI-452 core CPU and generates an interrupt or DMA request to either CPU, host or internal, as a FIFO service request.

The FIFO consists of two channels:the Input FIFO and the Output FIFO. The division of the FIFO module array, 128 bytes, between Input channel and Output channel is programmable by the user. Each FIFO byte has an additional logical ninth bit to distinguish between a data byte and a Data Stream Command byte. Additionally, Immediate Commands allow direct, interrupt driven, bi-directional communication between the UPI-452 internal CPU and external host CPU, bypassing the FIFO.

The on-chip DMA processor allows high speed data transfers from one writeable memory space to another. As many as 64K bytes can be transferred in a single DMA operation. Three distinct memory spaces may be used in DMA operations; Internal Data Memory, External Data Memory, and the Special Function Registers (including the FIFO IN, FIFO OUT, and Serial Channel Special Functions Registers).

September 1987
Order Number: 231428-003

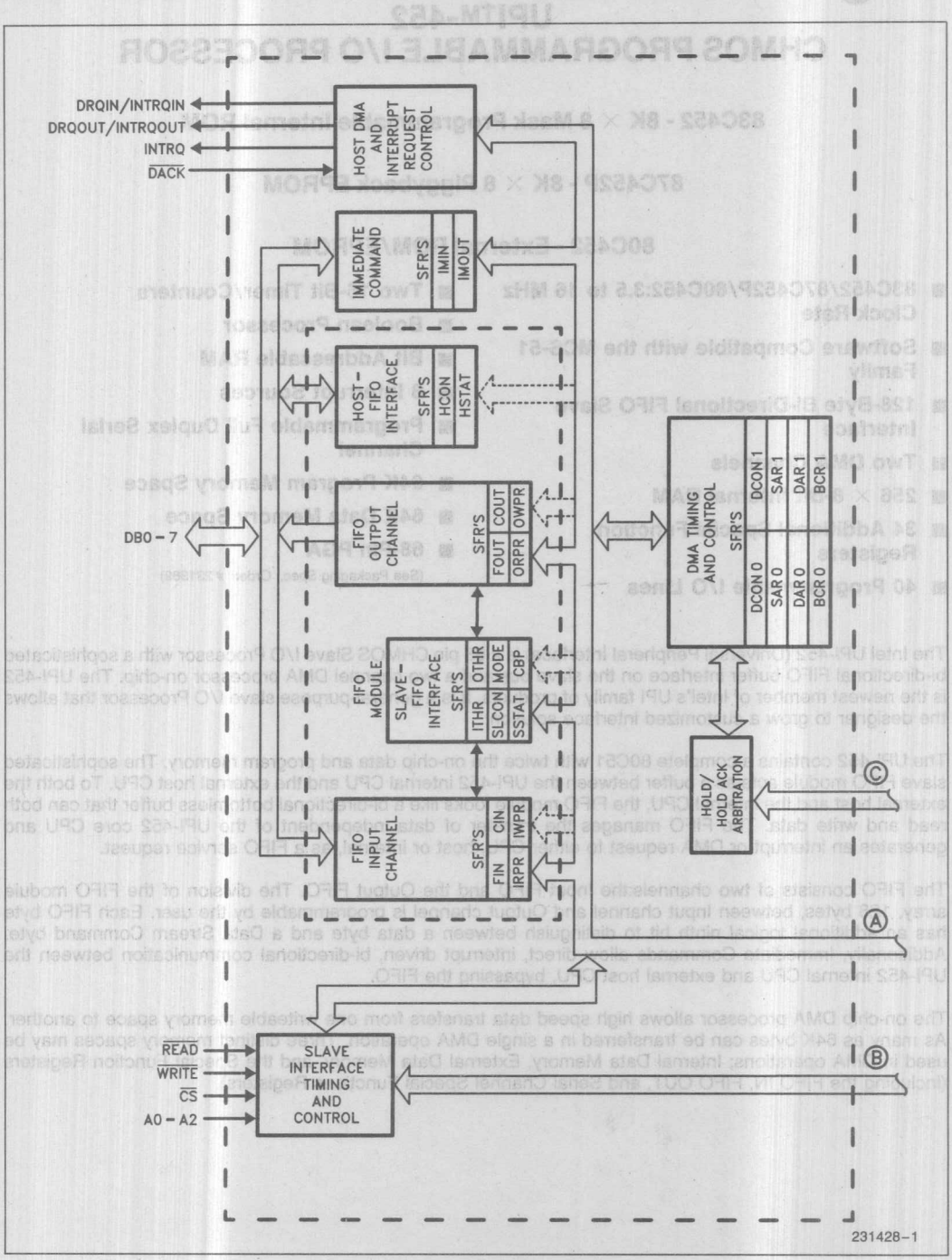


Figure 1. Architectural Block Diagram

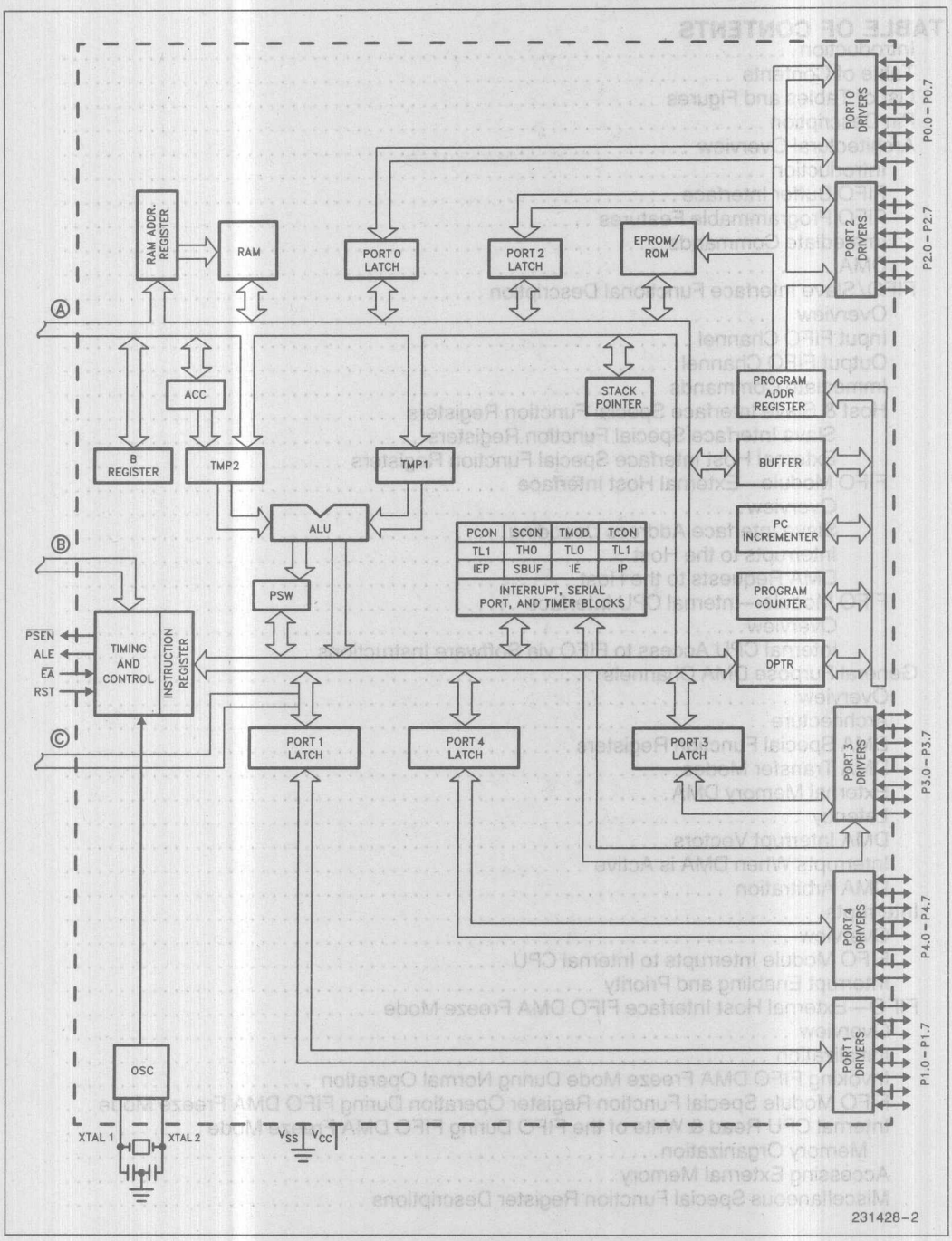


Figure 1. Architectural Block Diagram (Continued)

TABLE OF CONTENTS

- Introduction
- Table of Contents
- List of Tables and Figures
- Pin Description
- Architectural Overview
  - Introduction
  - FIFO Buffer Interface
  - FIFO Programmable Features
  - Immediate Commands
  - DMA
- FIFO/Slave Interface Functional Description
  - Overview
  - Input FIFO Channel
  - Output FIFO Channel
  - Immediate Commands
  - Host & Slave Interface Special Function Registers
    - Slave Interface Special Function Registers
    - External Host Interface Special Function Registers
  - FIFO Module—External Host Interface
    - Overview
    - Slave Interface Address Decoding
    - Interrupts to the Host
    - DMA Requests to the Host
  - FIFO Module—Internal CPU Interface
    - Overview
    - Internal CPU Access to FIFO via Software Instructions
- General Purpose DMA Channels
  - Overview
  - Architecture
  - DMA Special Function Registers
  - DMA Transfer Modes
  - External Memory DMA
  - Latency
  - DMA Interrupt Vectors
  - Interrupts When DMA is Active
  - DMA Arbitration
- Interrupts
  - Overview
  - FIFO Module Interrupts to Internal CPU
  - Interrupt Enabling and Priority
- FIFO—External Host Interface FIFO DMA Freeze Mode
  - Overview
  - Initialization
  - Invoking FIFO DMA Freeze Mode During Normal Operation
  - FIFO Module Special Function Register Operation During FIFO DMA Freeze Mode
  - Internal CPU Read & Write of the FIFO During FIFO DMA Freeze Mode
    - Memory Organization
  - Accessing External Memory
  - Miscellaneous Special Function Register Descriptions

LIST OF TABLES AND FIGURES

Figures:

1. Architectural Block Diagram
2. UPI-452 68-Pin PGA Pinout Diagram
3. UPI-452 Conceptual Block Diagram
4. UPI-452 Functional Block Diagram
5. Input FIFO Channel Functional Block Diagram
6. Output FIFO Channel Functional Block Diagram
7a. Handshake Mechanisms for Handling Immediate Command IN Flowchart
7b. Handshake Mechanisms for Handling Immediate Command OUT Flowchart
8. DMA Transfer from: External to External Memory
9. DMA Transfer from: External to Internal Memory
10. DMA Transfer from: Internal to External Memory
11. DMA Transfer Waveform: Internal to Internal Memory
12. Disabling FIFO to Host Slave Interface Timing Diagram

Tables:

1. Input FIFO Channel Registers
2. Output FIFO Channel Registers
3. UPI-452 Address Decoding
4. DMA Accessible Special Function Registers
5. DMA Mode Control - PCON SFR
6. Interrupt Priority
7. Interrupt Vector Addresses
8. Slave Bus Interface Status During FIFO DMA Freeze Mode
9. FIFO SFR's Characteristics During FIFO DMA Freeze Mode
10. Threshold SFRs Range of Values and Number of Bytes to be Transferred
11a. 80C51 Special Function Registers
11b. UPI-452 Additional Special Function Registers
12. Program Status Word (PSW)
13. PCON Special Function Register

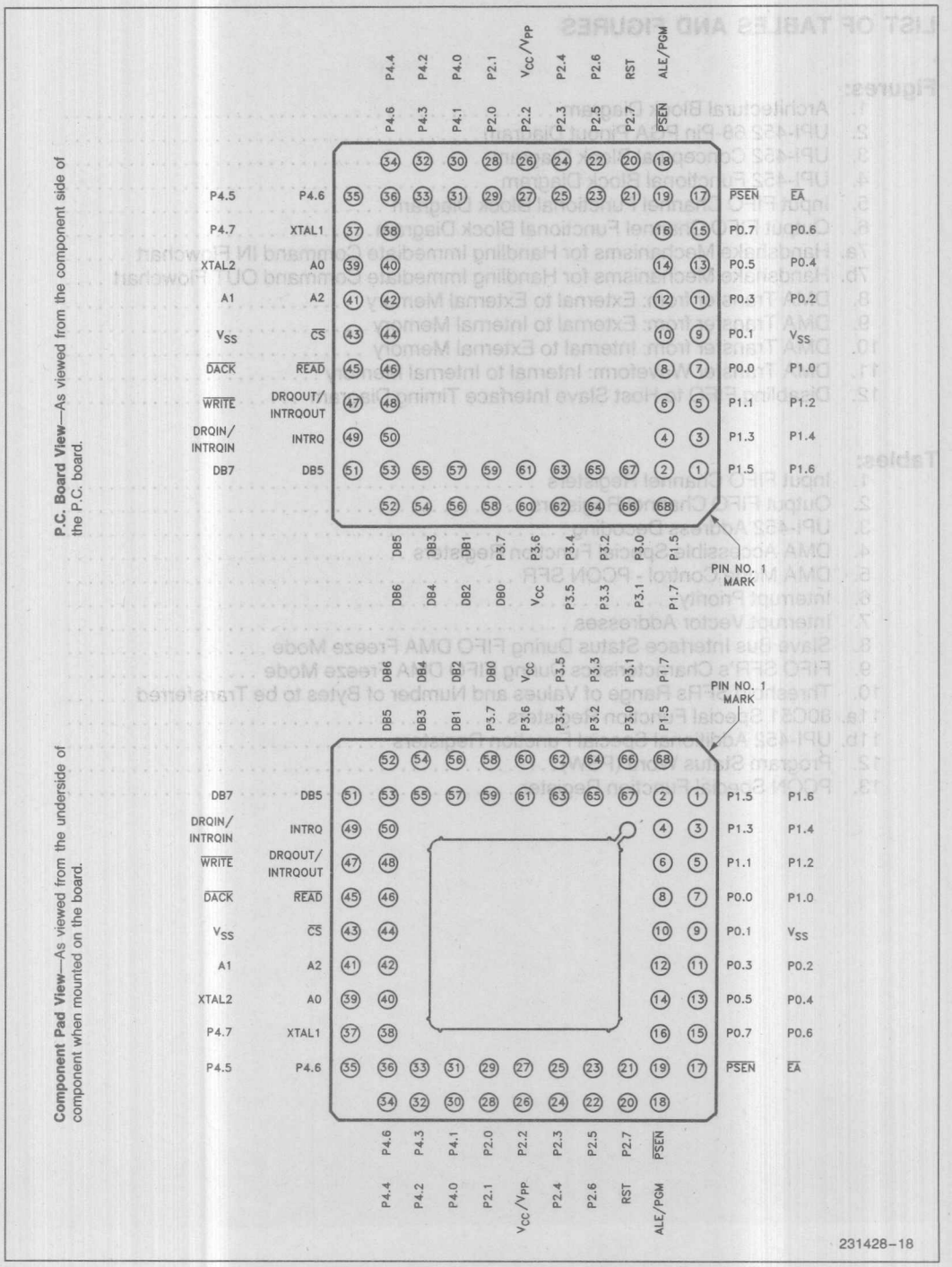


Figure 2. UPI-452 68-Pin PGA Pinout Diagram

UPI MICROCONTROLLER FAMILY

The UPI-452 joins the current members of the UPI microcontroller family. UPI's are derivatives of the MCS™ family of microcontrollers. Because of their on-chip system bus interface, UPI's are designed to be system bus "slaves", while their microcontroller counterparts are intended as system bus "masters".

These UPI Microcontrollers are fully supported by Intel's EPROM programmers (iUP-201) and development tools (ICE, ASM and PLM).

Packaging

The 80C452 comes in a 68-pin PGA (Pin Grid Array) package, while the 87C452P will be offered in a piggyback package. This piggyback package will consist of the standard 68-pin PGA package with a 2764A EPROM soldered on top. These two packages allow designers to use either on-chip EPROM or external memory for their initial designs. The 83C452 (ROM version) will come in the standard 68-pin PGA package. A complete description of 87C452P programming can be found at the end of this data sheet.

UPI Family (Slave Configuration)	MCS Family (Master Configuration)	Speed	RAM (Bytes)	ROM (Bytes)	EPROM (Bytes)
80C452	80C51	12 MHz	256	—	—
83C452	80C51	12 MHz	256	8K	—
87C452P	80C51	12 MHz	256	—	8K
80C452-1	80C51	16 MHz	256	—	—
83C452-1	80C51	16 MHz	256	8K	—
87C452P-1	80C51	16 MHz	256	—	8K

UPI-452 PIN DESCRIPTIONS

Symbol	Pin #	Type	Name and Function
V_{SS}	9/43	I	Circuit Ground.
V_{CC}	60	I	+5V power supply during normal, idle, programming and verification operation. It is also the standby power pin for power down mode.
XTAL1	38	I	Input to the oscillator's high gain amplifier. A crystal or external source can be used.
XTAL2	39	O	Output from the high gain amplifier.
Port 0 (AD0–AD7) P0.0 .1 .2 .3 .4 .5 .6 P0.7	 8 10 11 12 13 14 15 16	I/O	Port 0 is an 8-bit open drain bi-directional I/O port. It is used for data input and output during programming and verification. External pullups are required during program verification. Port 0 can sink eight LS TTL inputs. It is also the multiplexed low-order address and data local expansion bus during accesses to external memory.

UPI-452 PIN DESCRIPTIONS (Continued)

Symbol	Pin #	Type	Name and Function
Port 1 (A0–A7) (HLD, HLDA) P1.0 .1 .2 .3 .4 .5 .6 P1.7	 7 6 5 4 3 2 1 68	I/O	Port 1 is an 8-bit quasi-bi-directional I/O port. It is used for low-order address byte during programming and verification. Port 1 can sink four LS TTL inputs. The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise, the port pin is stuck at 0. Pins P1.5 and P1.6 are multiplexed with HLD and HLDA respectively whose functions are defined as below: Port Pin — Alternate Function P1.5 — $\overline{HLD}$ —Local bus hold input/output signal P1.6 — $\overline{HLDA}$ —Local bus hold acknowledge input
Port 2 (A8–A15) P2.0 .1 .2 .3 .4 .5 .6 .7	 29 28 27 25 24 23 22 21	I/O	Port 2 is an 8-bit quasi-bi-directional I/O port. It also emits the high-order 8 bits of address when accessing local expansion bus external memory (or during 87C452P programming and verification). Port 2 can sink four LS TTL inputs.
Port 3 P3.0 .1 .2 .3 .4 .5 .6 P3.7	 67 66 65 64 63 62 61 59	I/O	Port 3 is an 8-bit quasi-bi-directional I/O port. It is also multiplexed with the interrupt, timer, local serial channel, RD/ and WR/ functions that are used by various options. The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise, the port pin is stuck at 0. Port 3 can sink four LS TTL inputs. The alternate functions assigned to the pins of Port 3 are as follows: Port Pin — Alternate Function P3.0 — RxD — Serial input port P3.1 — TxD — Serial output port P3.2 — INT0 — Interrupt 0 Input P3.3 — INT1 — Interrupt 1 Input P3.4 — T0 — Input to counter 0 P3.5 — T1 — Input to counter 1 P3.6 — WR/ — The write control signal latches the data from Port 0 outputs into the External Data Memory on the local bus. P3.7 — RD/ — The read control signal latches the data from Port 0 outputs on the local bus.

UPI-452 PIN DESCRIPTIONS (Continued)

Symbol	Pin #	Type	Name and Function
Port 4 P4.0 .1 .2 .3 .4 .5 .6 .7	 30 31 32 33 34 35 36 37	I/O	Port 4 is an 8-bit quasi-bi-directional I/O port. Port 4 can sink/source four TTL inputs. It is also used as the control signals during EPROM programming and verification drive pins as follows: Port Pin — Alternate Function P4.5 — '1' during program and verify P4.6 — '0' during program and verify P4.7 — '0' during verify - used as output enable '1' during programming w/ ALE = 0 Note: see Programming and Verification Characteristics in AC/DC Specification section.
RST	20	I	A high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown resistor permits Power-on reset using only a capacitor connected to V_{CC}. This pin does not receive the power down voltage as is the case for HMOS MCS-51 family members. This function has been transferred to the V_{CC} pin.
ALE/PGM	18	I/O	Provides Address Latch Enable output used for latching the address into external memory during normal operation. Receives the program pulse input during EPROM programming. ALE can sink/source eight LS TTL inputs.
$\overline{PSEN}$	19	O	The Program Store Enable output is a control signal that enables the external Program Memory to the bus during normal fetch operation. $\overline{PSEN}$ can sink/source eight LS TTL inputs.
$\overline{EA}$	17	I	When held at TTL high level, the UPI-452 executes instructions from the internal ROM/EPROM when the PC is less than 8192 (8K, 2000H). When held at a TTL low level, the UPI-452 fetches all instructions from external Program Memory.
DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7	58 57 56 55 54 53 52 51	I/O	Host Bus Interface is an 8-bit bi-directional bus. It is used to transfer data and commands between the UPI-452 and the host processor. This bus can sink/source eight LS TTL inputs.
$\overline{CS}$	44	I	This pin is the Chip Select of the UPI-452.
A0 A1 A2	40 41 42	I	These three address lines are used to interface with the host system. They define the UPI-452 operations. The interface is compatible with the Intel microprocessors and the MULTIBUS.
$\overline{READ}$	46	I	This pin is the read strobe from the host CPU. Activating this pin causes the UPI-452 to place the contents of the Output FIFO (either a command or data) or the Host Status/Control Special Function Register on the Slave Data Bus.
$\overline{WRITE}$	47	I	This pin is the write strobe from the host. Activating this pin will cause the value on the Slave Data Bus to be written into the register specified by A0–A2.
DRQIN/ INTRQIN	49	O	This pin requests an input transfer from the host system whenever the Input Channel requires data.
DRQOUT/ INTRQOUT	48	O	This output pin requests an output transfer whenever the Output Channel requires service. If the external host to UPI-452 DMA is enabled, and a Data Stream Command is at the Output FIFO, DRQOUT is deactivated and INTRQ is activated (see 'GENERAL PURPOSE DMA CHANNELS' section).

UPI-452 PIN DESCRIPTIONS (Continued)

Symbol	Pin #	Type	Name and Function
INTRQ	50	O	This output pin is used to interrupt the host processor when an Immediate Command Out or an error condition is encountered. It is also used to interrupt the host processor when the FIFO requests service if the DMA is disabled and INTRQIN and INTRQOUT are not used.
$\overline{DACK}$	45	I	This pin is the DMA acknowledge for the host bus interface Input and Output Channels. When activated, a write command will cause the data on the Slave Data Bus to be written as data to the Input Channel (to the Input FIFO). A read command will cause the Output Channel to output data (from the Output FIFO) on to the Slave Data Bus. This pin should be driven high (+5V) in systems which do not have a DMA controller (see Address Decoding).
V_{CC}/V_{PP}	26	I	+5V power supply during operation. The V_{CC} pin receives the +12V EPROM programming and verification supply voltage.

ARCHITECTURAL OVERVIEW

Introduction

The UPI-452 slave microcontroller incorporates an 80C51 with double the program and data memory, a slave interface which allows it to be connected directly to the host system bus as a peripheral, a FIFO buffer module, a two channel DMA processor, and a fifth I/O port (Figure 3). The UPI-452 retains all of the 80C51 architecture, and is fully compatible with the MCS-51 instruction set.

The Special Function Register (SFR) interface concept introduced in the MCS-51 family of microcontrollers has been expanded in the UPI-452. To the 20 Special Function Registers of the MCS-51, the UPI-452 adds 34 more. These additional Special Function Registers, like those of the MCS-51, provide access to the UPI-452 functional elements including the FIFO, DMA and added interrupt capabilities. Several of the 80C51 core Special Function Registers have also been expanded to support added features of the UPI-452.

This data sheet describes the unique features of the UPI-452. Refer to the 80C51 data sheet for a description of the UPI-452's core CPU functional blocks including;

- Timers/Counters
- I/O Ports
- Interrupt timing and control (other than FIFO and DMA interrupts)
- Serial Channel
- Local Expansion Bus
- Program/Data Memory structure
- Power-Saving Modes of Operation *
- CHMOS Features
- Instruction Set

* except 87C452P piggyback package

Figure 3 contains a conceptual block diagram of the UPI-452. Figure 4 provides a functional block diagram.

FIFO Buffer Interface

A unique feature of the UPI-452 is the incorporation of a 128 byte FIFO array at the host-slave interface. The FIFO allows asynchronous bi-directional transfers between the host CPU and the internal CPU.

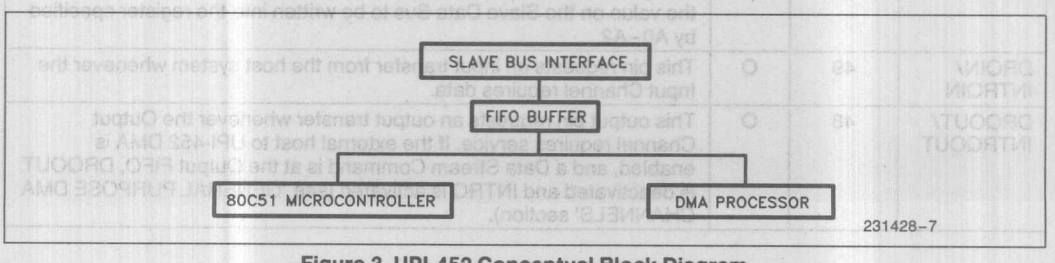


Figure 3. UPI-452 Conceptual Block Diagram

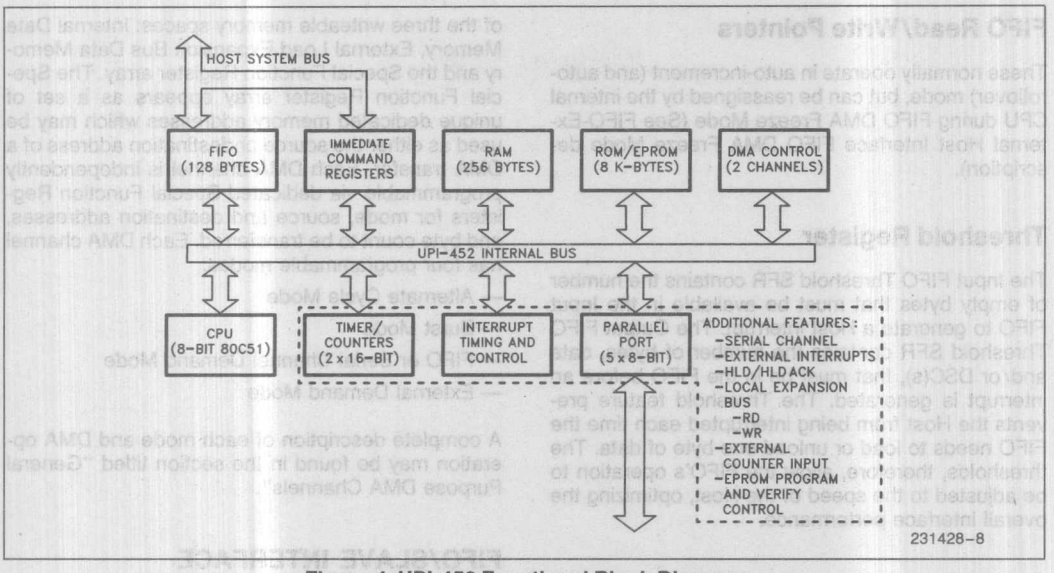


Figure 4. UPI-452 Functional Block Diagram

The division of the 128 bytes between Input and Output channels is user programmable allowing maximum flexibility. If the entire 128 byte FIFO is allocated to the Input channel, a high performance Host can transfer up to 128 bytes at one time, then dedicate its resources to other functions while the internal CPU processes the data in the FIFO. Various handshake signals allow the external Host to operate independently and without frequent monitoring of the UPI-452 internal CPU. The FIFO Buffer insures that the slave processor receives data in the same order that it was sent by the host without the need to keep track of addresses. Three slave bus interface handshake methods are supported by the UPI-452: DMA, Interrupt and Polled.

The FIFO is nine bits wide. The ninth bit acts as a command/data flag. Commands written to the FIFO by either the host or internal CPU are called Data Stream Commands or DSCs. DSCs are written to the input FIFO by the Host via a unique external address. DSCs are written to the output FIFO by the internal CPU via the COMMAND OUT Special Function Register (SFR). When encountered by the host or internal CPU a Data Stream Command can be used as an address vector to user defined service routines. DSCs provide synchronization of data and commands between the Host and internal CPU.

FIFO PROGRAMMABLE FEATURES

Size of Input/Output Channels

The 128 bytes of FIFO space can be allocated between the Input and Output channels via the Channel Boundary Pointer (CBP) SFR. This register contains the number of address locations assigned to the Input channel. The remaining address locations are automatically assigned to the Output FIFO. The CBP SFR can only be programmed by the internal CPU during FIFO DMA Freeze Mode (See FIFO-External Host Interface FIFO DMA Freeze Mode description). The CBP is initialized to 40H (64 bytes) upon reset.

The number in the Channel Boundary Pointer SFR is actually the first address location of the Output FIFO. Writing to the CBP SFR reassigns the Input and Output FIFO address space. Whenever the CBP is written, the Input FIFO pointers are reset to zero and the Output FIFO pointers are set to the value in the CBP SFR.

All of the FIFO space may be assigned to one channel. In such a situation the other channel's data path consists of a single SFR (FIFO IN/COMMAND IN or FIFO OUT/COMMAND OUT SFR) location.

CBP Register	Input FIFO Size	Output FIFO Size
0	1	128
1	1	128
2	2	126
3	3	125
4	4	124
•	•	•
7B	123	5
7C	124	4
7D	125	3
7E	128	1
7F	128	1

FIFO Read/Write Pointers

These normally operate in auto-increment (and auto-rollover) mode, but can be reassigned by the internal CPU during FIFO DMA Freeze Mode (See FIFO-External Host Interface FIFO DMA Freeze Mode description).

Threshold Register

The Input FIFO Threshold SFR contains the number of empty bytes that must be available in the Input FIFO to generate a Host interrupt. The Output FIFO Threshold SFR contains the number of bytes, data and/or DSC(s), that must be in the FIFO before an interrupt is generated. The Threshold feature prevents the Host from being interrupted each time the FIFO needs to load or unload one byte of data. The thresholds, therefore, allow the FIFO's operation to be adjusted to the speed of the Host, optimizing the overall interface performance.

Immediate Commands

The UPI-452 provides, in addition to data and DSCs, a third direct means of communication between the external Host and internal CPU called Immediate Commands. As the name implies, an Immediate Command is available to the receiving CPU immediately, via an interrupt, without being entered into the FIFO as are Data Stream Commands. Like Data Stream Commands, Immediate Commands are written either via a unique external address by the host CPU, or via dedicated SFR by the internal CPU.

The DSC and/or Immediate Command interface may be defined as either Interrupt or Polled under user program control via the Interrupt Enable (IE), Slave Control Register (SLCON), and Interrupt Enable Priority (IEP) Special Function Registers, for the internal CPU and via the Host Control SFR for the external Host CPU.

DMA

The UPI-452 contains a two channel internal DMA controller which allows transfer of data between any of the three writeable memory spaces: Internal Data Memory, External Load Expansion Bus Data Memory and the Special Function Register array. The Special Function Register array appears as a set of unique dedicated memory addresses which may be used as either the source or destination address of a DMA transfer. Each DMA channel is independently programmable via dedicated Special Function Registers for mode, source and destination addresses, and byte count to be transferred. Each DMA channel has four programmable modes:

— Alternate Cycle Mode

— Burst Mode

— FIFO or Serial Channel Demand Mode

— External Demand Mode

A complete description of each mode and DMA operation may be found in the section titled "General Purpose DMA Channels".

FIFO/SLAVE INTERFACE FUNCTIONAL DESCRIPTION

Overview

The FIFO is a 128 Byte RAM array with recirculating pointers to manage the read and write accesses. The FIFO consists of an Input and an Output channel. Access cycles to the FIFO by the internal CPU and external Host are interleaved and appear to be occurring concurrently to both the internal CPU and external Host. Interleaving access cycles ensures efficient use of this shared resource. The internal CPU accesses the FIFO in the same way it would access any of the Special Function Registers e.g., direct and register indirect addressing as well as arithmetic and logical instructions.

Input FIFO Channel

The Input FIFO Channel provides for data transfer from the external Host to the internal CPU (Figure 5). The registers associated with the Input Channel during normal operation are listed in Table 1*.

Table 1. Input FIFO Channel Registers*

	Register Name	Description
1)	Input Buffer Latch	Host CPU Write only
2)	FIFO IN SFR	Internal CPU Read only
3)	COMMAND IN SFR	Internal CPU Read only
4)	Input FIFO Read Pointer SFR	Internal CPU Read only
5)	Input FIFO Write Pointer SFR	Internal CPU Read only
6)	Input FIFO Threshold SFR	Internal CPU Read only

*See "FIFO-EXTERNAL HOST INTERFACE FIFO DMA FREEZE MODE" section for FIFO DMA Freeze Mode SFR characteristics description.

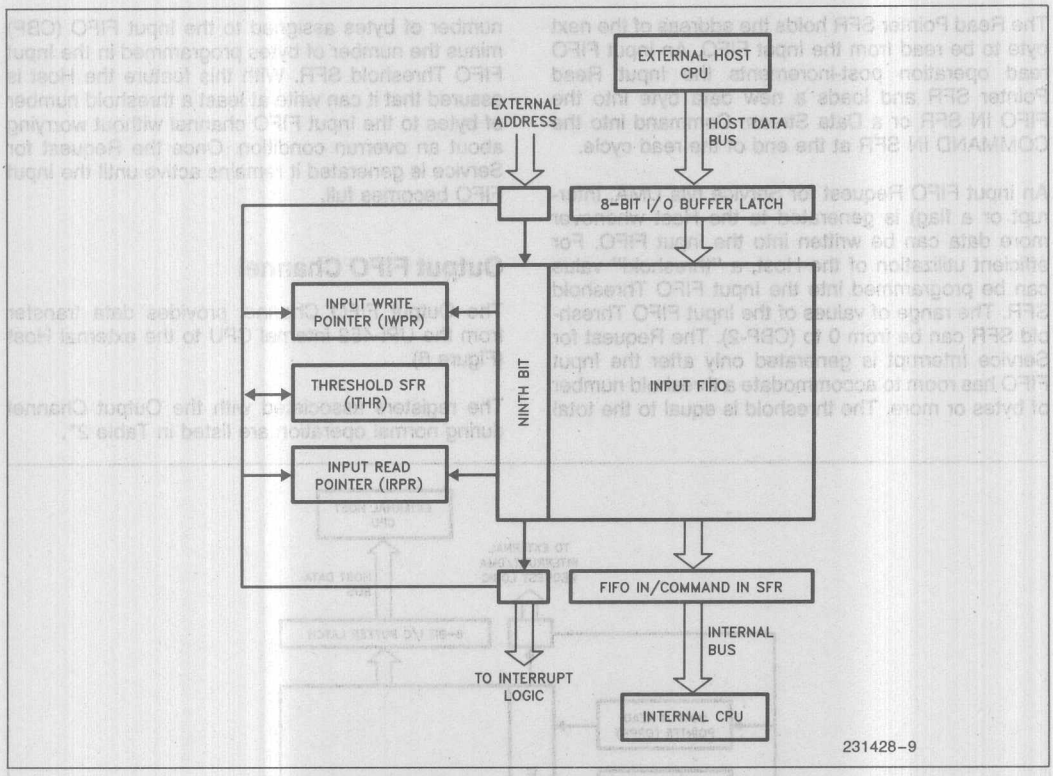


Figure 5. Input FIFO Channel Functional Block Diagram

The host CPU writes data and Data Stream Commands into the Input Buffer Latch on the rising edge of the external WR signal. External addressing determines whether the byte is a data byte or Data Stream Command and the FIFO logic sets the ninth bit of the FIFO accordingly as the byte is moved from the Input Buffer Latch into the FIFO. A "1" in the ninth bit indicates that the incoming byte is a Data Stream Command. The internal CPU reads data bytes via the FIFO IN SFR, and Data Stream Commands via the COMMAND IN SFR.

A Data Stream Command will generate an interrupt to the internal CPU prior to being read and after completion of the previous operation. The DSC can then be read via the COMMAND IN SFR. Data can only be read via the FIFO IN SFR and Data Stream Commands via the COMMAND IN SFR. Attempting to read Data Stream Commands as data by addressing the FIFO IN SFR will result in "0FFH" being read, and the Input FIFO Read Pointer will remain intact. (This prevents accidental misreading of Data Stream Commands.) Attempting to read data as Data Stream Commands will have the same consequence.

The Input FIFO Channel addressing is controlled by the Input FIFO Read and Write Pointer SFRs. These SFRs are read only registers during normal operation. However, during FIFO DMA Freeze Mode (See FIFO-External Host Interface FIFO DMA Freeze Mode description), the internal CPU has write access to them. Any write to these registers in normal mode will have no effect. The Input Write Pointer SFR contains the address location to which data/commands are written from the Input Buffer Latch. The write pointer is automatically incremented after each write and is reset to zero if equal to the CBP, as the Input FIFO operates as a circular buffer.

If a write is performed on an empty FIFO, the first byte is also written into the FIFO IN or COMMAND IN SFR. If the Host continues writing while the Input FIFO is full, an external interrupt, if enabled, is sent to the host to signal the overrun condition. The writes are ignored by the FIFO control logic. Similarly, an internal CPU read of an empty FIFO will cause an underrun error interrupt to be generated to the internal CPU and a value of "0FFH" will be read by the internal CPU.

The Read Pointer SFR holds the address of the next byte to be read from the Input FIFO. An Input FIFO read operation post-increments the Input Read Pointer SFR and loads a new data byte into the FIFO IN SFR or a Data Stream Command into the COMMAND IN SFR at the end of the read cycle.

An Input FIFO Request for Service (via DMA, Interrupt or a flag) is generated to the Host whenever more data can be written into the Input FIFO. For efficient utilization of the Host, a "threshold" value can be programmed into the Input FIFO Threshold SFR. The range of values of the Input FIFO Threshold SFR can be from 0 to (CBP-2). The Request for Service Interrupt is generated only after the Input FIFO has room to accommodate a threshold number of bytes or more. The threshold is equal to the total number of bytes assigned to the Input FIFO (CBP) minus the number of bytes programmed in the Input FIFO Threshold SFR. With this feature the Host is assured that it can write at least a threshold number of bytes to the Input FIFO channel without worrying about an overrun condition. Once the Request for Service is generated it remains active until the Input FIFO becomes full.

Output FIFO Channel

The Output FIFO Channel provides data transfer from the UPI-452 internal CPU to the external Host (Figure 6).

The registers associated with the Output Channel during normal operation are listed in Table 2*.

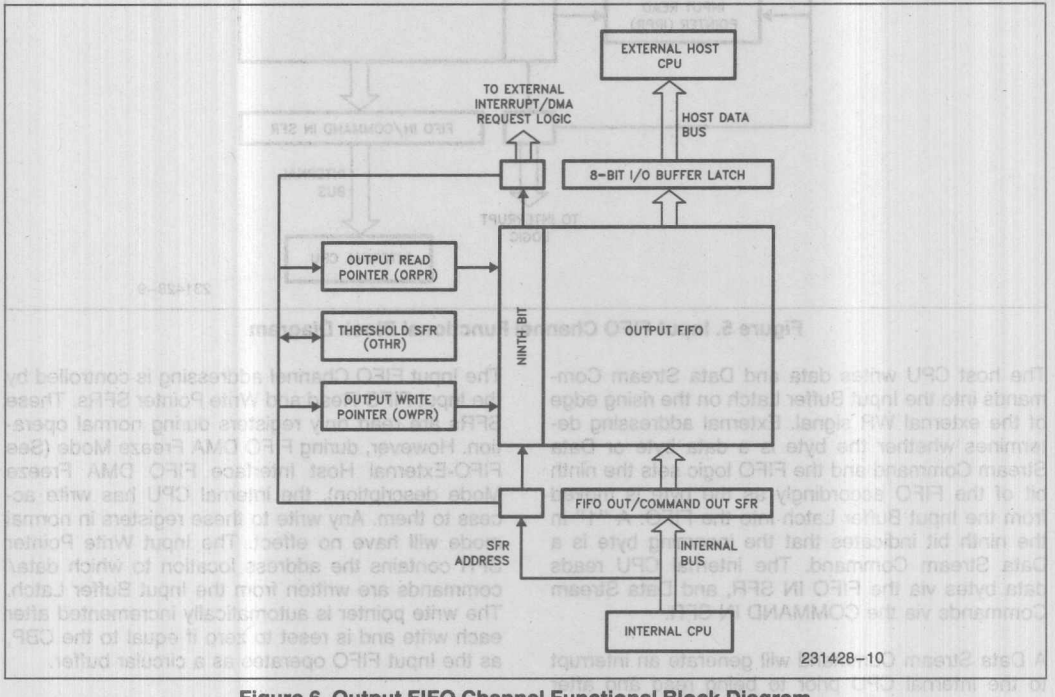


Figure 6. Output FIFO Channel Functional Block Diagram

Table 2. Output FIFO Channel Registers

	Register Name	Description
1)	Output Buffer Latch	Host CPU Read only
2)	FIFO OUT SFR	Internal CPU Read and Write
3)	COMMAND OUT SFR	Internal CPU Read and Write
4)	Output FIFO Read Pointer SFR	Internal CPU Read only
5)	Output FIFO Write Pointer SFR	Internal CPU Read only
6)	Output FIFO Threshold SFR	Internal CPU Read only

*See "FIFO-EXTERNAL HOST INTERFACE FIFO DMA FREEZE MODE" section for FIFO DMA Freeze Mode register characteristics description.

The UPI-452 internal CPU transfers data to the Output FIFO via the FIFO OUT SFR and commands via the COMMAND OUT SFR. If the byte is written to the COMMAND OUT SFR, the ninth bit is automatically set (=1) to indicate a Data Stream Command. If the byte is written to the FIFO OUT SFR the ninth bit is cleared (=0). Thus the FIFO OUT and COMMAND OUT SFRs are the same but the address determines whether the byte entered in the FIFO is a DSC or data byte.

The Output FIFO preloads a byte into the Output Buffer Latch. When the Host issues a RD/ signal, the data is immediately read from the Output Buffer Latch. The next data byte is then loaded into the Output Buffer Latch, a flag is set and an interrupt, if enabled, is generated if the byte is a DSC (ninth bit is set). The operation is carefully timed such that an interrupt can be generated in time for it to be recognized by the Host before its next read instruction. Internal CPU write and external Host read operations are interleaved at the FIFO so that they appear to be occurring concurrently.

The Output FIFO read and write pointer operation is the same as for the Input Channel. Writing to the FIFO OUT or COMMAND OUT SFRs will increment the Output Write Pointer SFR but reading from it will leave the write pointer unchanged. A rollover of the Output FIFO Write Pointer causes the pointer to be reset to the value in the Channel Boundary Pointer (CBP) SFR.

If the external host attempts to read a Data Stream Command as a data byte it will result in invalid data (0FFH) being read. The DSC is not lost because the invalid read does not increment the pointer. Similarly attempting to read a data byte as a Data Stream Command has the same result.

A Request for Service is generated to the external Host under the following two conditions:

1.) Whenever the internal CPU has written a threshold number of bytes or more into the Output FIFO (threshold = (OTHR) + 1). The threshold number should be chosen such that the bus latency time for the external Host does not result in a FIFO overrun error condition on the internal CPU side. The threshold limit should be large enough to make a bus request by the UPI-452 to the external host CPU worthwhile. Once a request for service is generated, the request remains active until the Output FIFO becomes empty. The range of values of the FIFO Output Threshold (OTHR) SFR is from 1 to the Output FIFO Size. The threshold number can be programmed via the OTHR SFR.

2.) The second type of Request for Service is called "Flush Mode" and occurs when the internal CPU writes a Data Stream Command into the Output FIFO. Its purpose is to ensure that a data block entered into the Output FIFO, which is less than the programmed threshold, will generate a Request for Service interrupt, if enabled, and be read, or "Flushed" from the Output FIFO, by the external host CPU regardless of the status of the OTHR SFR.

Immediate Commands

Immediate Commands provide direct communication between the external Host and UPI-452. Unlike Data Stream Commands which are entered into the FIFO, the Immediate Command is available to the receiving CPU directly, bypassing the FIFO. The Immediate Command can serve as a program vector pointing into a jump table in the recipients software. Immediate Command Interrupts are generated, if enabled, and a bit in the appropriate Status Register is set when an Immediate Command is input or output. A similar bit is provided to acknowledge when an Immediate Command has been read and whether the register is available to receive another command. The bits are reset when the Immediate Commands are read. Two Special Function Registers are dedicated to the Immediate Command interface. External addressing determines whether the Host is accessing the Input FIFO or the Immediate Command IN (IMIN) SFR. The internal CPU writes Immediate Commands to the Immediate Command OUT (IMOUT) SFR.

Both processors have the ability to enable or disable Immediate Command Interrupts. By disabling the interrupt, the recipient of the Immediate Command can poll the status SFR and read the Immediate Command at its convenience. Immediate Commands should only be written when the appropriate Immediate Command SFR is empty (as indicated in the appropriate status SFR:HSTAT/SSTAT). Similarly, the Immediate Command SFR should only be read when there is data in the Register.

The flowcharts in Figure 7a and 7b illustrate the proper handshake mechanisms between the external Host and internal CPU when handling Immediate Commands.

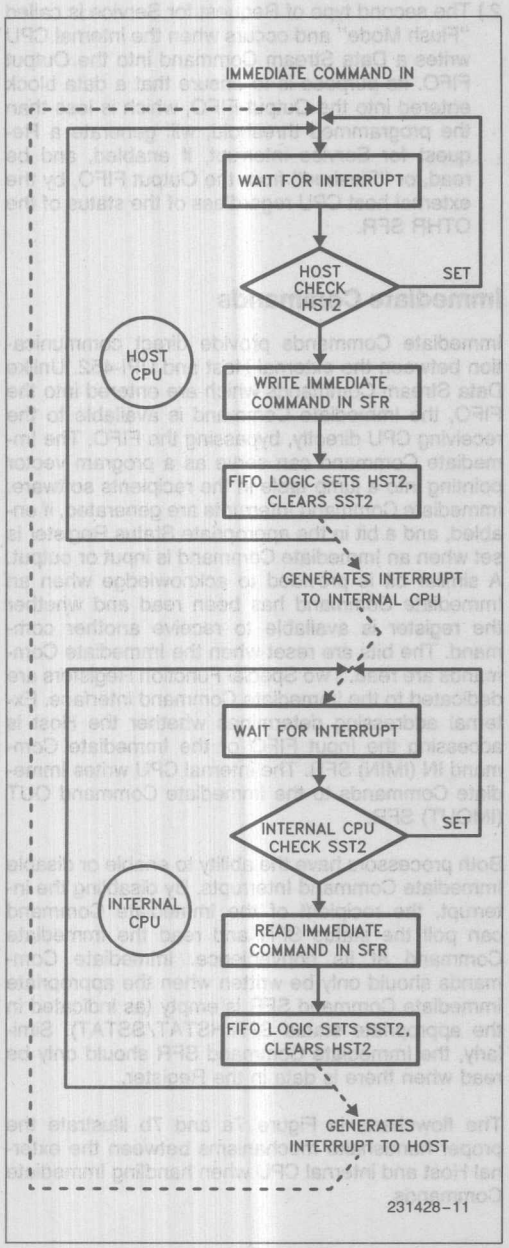


Figure 7a. Handshake Mechanisms for Handling Immediate Command IN Flowchart

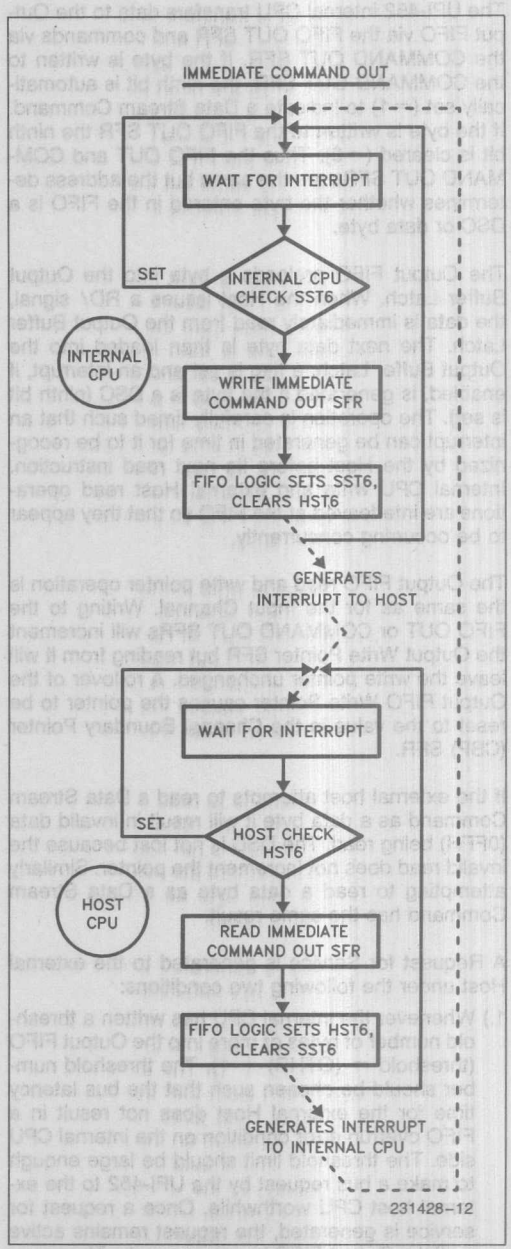


Figure 7b. Handshake Mechanisms for Handling Immediate Command OUT Flowchart

HOST & SLAVE INTERFACE SPECIAL FUNCTION REGISTERS

Slave Interface Special Function Registers

The Internal CPU interfaces with the FIFO slave module via the following registers:

1) Mode Special Function Register (MODE)
2) Slave Control Special Function Register (SLCON)
3) Slave Status Special Function Register (SSTAT)

Each register resides in the SFR Array and is accessible via all direct addressing modes except bit. Only the Slave Control Register (SLCON) is bit addressable.

1) MODE Special Function Register (MODE)

The MODE SFR provides the primary control of the external host-FIFO interface. It is included in the SFR Array so that the internal CPU can configure the external host-FIFO interface should the user decide that the UPI-452 slave initialize itself independent of the external host CPU.

The MODE SFR can be directly modified by the internal CPU through direct address instructions. It can also be indirectly modified by the external host CPU by setting up a MODE SFR service routine in the UPI-452 program memory and having the host issue a Command, either Immediate or DSC, to vector to that routine.

Symbolic Address									Physical Address
MODE	—	MD6	MD5	MD4	—	—	—	—	0F9H
	(MSB)							(LSB)	
	Status On Reset:								
	1*	0	0	0	1*	1*	1*	1*	

MD7 (reserved)**

MD6 Request for Service to external CPU via;

1 = DMA (DRQIN/DRQOUT) request to external host when the Input or Output FIFO channel requests service

0 = Interrupt (INTRQIN/INTRQOUT or INTRQ) to external host when the Input or Output FIFO channel requests service or a DSC is encountered in the I/O Buffer Latch

MD5 Configure DRQIN/INTRQIN and DRQOUT/INTRQOUT to be either;

1 = Enable (Actively driven)

0 = Disable (Tri-state)

MD4 Configure INTRQ to be either;

1 = Enable (Actively driven)

0 = Disable (Tri-state)

MD3 (reserved) **

MD2 (reserved) **

MD1 (reserved) **

MD0 (reserved) **

2) Slave Control SFR (SLCON)

The Slave Control SFR is used to configure the FIFO-internal CPU interface. All interrupts are to the internal CPU.

Symbolic Address									Physical Address
SLCON	IFI	OFI	ICII	ICOI	FRZ	—	IFRS	OFRS	0E8H
	(MSB)							(LSB)	
Status On Reset:	0	0	0	0	0	1*	0	0	

IFI Enable Input FIFO Interrupt (due to Underrun Error Condition, Data Stream Command or Request Service)
1 = Enable
0 = Disable

OFI Enable Output FIFO Interrupt (due to Overrun Error Condition or Request Service)
1 = Enable
0 = Disable
Note: If the DMA is configured to service a FIFO demand, then the Request for Service Interrupt is not generated.

ICII Generate Interrupt when a command is written to the Immediate Command in Register
1 = Enable
0 = Disable

ICOI Generate Interrupt when Immediate Command Out Register is Available
1 = Enable
0 = Disable

FRZ Enable FIFO DMA Freeze Mode
1 = Normal operation
0 = FIFO DMA Freeze Mode

SC2 (reserved) **

IFRS Input FIFO Channel Request for Service
1 = Request when Input FIFO not empty
0 = Request when Input FIFO full

OFRS Output FIFO Channel Request for Service
1 = Request when Output FIFO not full
0 = Channel Request when Output FIFO empty

NOTES:
*A '1' will be read from all SFR reserved locations except HCON SFR, HC0 and HC2.
**'reserved'—these locations are reserved for future use by Intel Corporation.

3) Slave Status SFR (SSTAT)

The bits in the Slave Status SFR reflect the status of the FIFO-internal CPU interface. It can be read during an internal interrupt service routine to determine the nature of the interrupt or read during a polling sequence to determine a course of action.

Symbolic Address									Physical Address
SSTAT	SST7	SST6	SST5	SST4	SST3	SST2	SST1	SST0	0E9H
	← Output FIFO Status →				← Input FIFO Status →				
Status On Reset:	1	0	0	0	1	1	1	1	
	(MSB)							(LSB)	

SST7 Output FIFO Overrun Error Condition
- 1 = No Error
- 0 = Error (latched until Slave Status SFR is read)

SST6 Immediate Command Out Register Status
- 1 = Full (i.e. Host CPU has not read previous Immediate Command Out sent by internal CPU)
- 0 = Available

SST5 FIFO DMA Freeze Mode Status
- 1 = Normal Operation
- 0 = FIFO DMA Freeze Mode in Progress

SST4 Output FIFO Request for Service Flag
- 1 = Output FIFO does not request service
- 0 = Output FIFO requests service

SST3 Input FIFO Underrun Error Condition Flag
- 1 = No Underrun Error
- 0 = Underrun Error (latched until Slave Status SFR is read)

SST2 Immediate Command In SFR Status
- 1 = Empty
- 0 = Immediate Command received from host CPU

SST1 Data Stream Command/Data at Input FIFO Flag
- 1 = Data (not DSC)
- 0 = DSC (at COMMAND IN SFR)

SST0 Input FIFO Request For Service Flag
- 1 = Input FIFO Does Not Request Service
- 0 = Input FIFO Request for Service

EXTERNAL HOST INTERFACE SPECIAL FUNCTION REGISTERS

The external host CPU has direct access to the following SFRs:

1) Host Control Special Function Register
2) Host Status Special Function Register

It can also access other SFRs by commanding the internal CPU to change them accordingly via Data Stream Commands or Immediate Commands. The protocol for implementing this is entirely determined by the user.

1) Host Control SFR (HCON)

By writing to the Host Control SFR, the host can enable or disable FIFO interrupts and DMA requests and can reset the UPI-452.

Symbolic Address									Physical Address
HCON	HC7	HC6	HC5	HC4	HC3	—	HC1	—	0E7H
	(MSB)							(LSB)	
	Status On Reset:								
	0	0	0	0	0	0*	0	0*	

HC7 Enable Output FIFO Interrupt due to Underrun Error Condition, Data Stream Command or Service Request
- 1 = Enable
- 0 = Disable

HC6 Enable Input FIFO Interrupt due to Overrun Error Condition, or Service Request
- 1 = Enable
- 0 = Disable

HC5 Enable the generation of the Interrupt due to Immediate Command Out being present
- 1 = Enable
- 0 = Disable

HC4 Enable the Interrupt due to the Immediate Command In Register being Available for a new Immediate Command byte
- 1 = Enable
- 0 = Disable

HC3 Reset UPI-452
- 1 = Software RESET
- 0 = Normal Operation

HC2 (reserved) **

HC1 Select between INTRQ and INTRQIN/INTRQOUT as Request for Service interrupt signal when DMA is disabled
- 1 = INTRQ
- 0 = INTRQIN or INTRQOUT

HC0 (reserved) **

NOTES:
*A '1' will be read from all SFR reserved locations except HCON SFR, HC0 and HC2.
**'reserved'—these locations are reserved for future use by Intel Corporation.

2) Host Status SFR (HSTAT)

The Host Status SFR provides information on the FIFO-Host Interface and can be used to determine the source of an external interrupt during polling. Like the Slave Status SFR, the Host Status SFR reflects the current status of the FIFO-external host interface.

Symbolic Address									Physical Address
HSTAT	HST7	HST6	HST5	HST4	HST3	HST2	HST1	HST0	0E6H
	← Output FIFO Status →				← Input FIFO Status →				
Status On Reset:									
	1	1	1	1	1	1/0*	1	1	
	(MSB)							(LSB)	

HST7 Output FIFO Underrun Error Condition
1 = No Underrun Error
0 = Underrun Error (latched until Host Status Register is read)

HST6 Immediate Command Out SFR Status
1 = Empty
0 = Immediate Command Present

HST5 Data Stream Command/Data at Output FIFO Status
1 = Data (not DSC)
0 = DSC (present at Output FIFO COMMAND OUT SFR)
(Note: Only if HST4 = 0, if HST4 = 1 then undetermined)

HST4 Output FIFO Request for Service Status
1 = No Request for Service
0 = Output FIFO Request for Service due to:

a. Output FIFO containing the threshold number of bytes or more

b. Internal CPU sending a block of data terminated by a DSC (DSC Flush Mode)

HST3 Input FIFO Overrun Error Condition
1 = No Overrun Error
0 = Overrun Error (latched until Host Status Register is read)

HST2 Immediate Command In SFR Status
1 = Full (i.e. Internal CPU has not read previous Immediate Command sent by Host)
0 = Empty

* Reset value;

'1' — if read by the external Host

'0' — if read by internal CPU (reads shadow latch - see FIFO DMA Freeze Mode description)

HST1 FIFO DMA Freeze Mode Status
1 = Freeze Mode in progress.
(In Freeze Mode, the bits of the Host Status SFR are forced to a '1' initially to prevent the external Host from attempting to access the FIFO. The definition of the Host Status SFR bits during FIFO DMA Freeze Mode can be found in FIFO DMA Freeze Mode description)
0 = Normal Operation

HST0 Input FIFO Request Service Status
1 = Input FIFO does not request service
0 = Input FIFO request service due to the Input FIFO containing enough space for the host to write the threshold number of bytes or more

FIFO MODULE - EXTERNAL HOST INTERFACE

Overview

The FIFO-external Host interface supports high speed asynchronous bi-directional 8-bit data transfers. The host interface is fully compatible with Intel microprocessor local busses and with MULTIBUS. The FIFO has two specialized DMA request pins for Input and Output FIFO channel DMA requests. These are multiplexed to provide a dedicated Request for Service interrupt (DRQIN/INTRQIN, DRQOUT/INTRQOUT).

The external Host can program, under user defined protocol, thresholds into the FIFO Input and Output Threshold SFRs which determine when the FIFO Request for Service interrupt is generated to the Host CPU. The FIFO module external Host interface is configured by the internal CPU via the MODE SFR. "The external Host can enable and disable Host interface interrupts via the Host Control SFR." Data Stream Commands in the Input FIFO channel allow the Host to influence the processing of data blocks and are sent with the data flow to maintain synchronization. Data Stream Commands in the Output FIFO Channel allow the internal CPU to perform the same function, and also to set the Output FIFO Request Service status logic to the host CPU regardless of the programmed value in the Threshold SFR.

Slave Interface Address Decoding

The UPI-452 determines the desired Host function through address decoding. The lower three bits of the address as well as the READ, WRITE, Chip Select ($\overline{CS}$) and DMA Acknowledge ($\overline{DACK}$) are used for decoding. Table 3 shows the pin states and the Read or Write operations associated with each configuration.

Interrupts to the Host

The UPI-452 interrupts the external Host via the INTRQ pin. In addition, the DRQIN and DRQOUT pins can be multiplexed as interrupt request lines, INTRQIN and INTRQOUT respectively, when DMA is disabled. This provides two special FIFO "Request for Service" interrupts.

There are eight FIFO-related interrupt sources; two from The Input FIFO; three from The Output FIFO; one from the Immediate Command Out SFR; one from the Immediate Command IN SFR; and one due to FIFO DMA Freeze Mode.

INPUT FIFO: The Input FIFO interrupt is generated whenever:

a. The Input FIFO contains space for a threshold number of bytes.

Table 3. UPI-452 Address Decoding

$\overline{DACK}$	$\overline{CS}$	A2	A1	A0	Read	Write
1	1	X	X	X	No Operation	No Operation
1	0	0	0	0	Data or DMA from Output FIFO Channel	Data or DMA to Input FIFO Channel
1	0	0	0	1	Data Stream Command from Output FIFO Channel	Data Stream Command to Input FIFO Channel
1	0	0	1	0	Host Status SFR Read	Reserved
1	0	0	1	1	Host Control SFR Read	Host Control SFR Write
1	0	1	0	0	Immediate Command SFR Read	Immediate Command to SFR Write
1	0	1	1	X	Reserved	Reserved
0	X	X	X	X	DMA Data from Output FIFO Channel	DMA Data to Input FIFO Channel
1	0	1	0	1	Reserved	Reserved

NOTES:
1. Attempting to read a DSC as a data byte will result in invalid data being read. The read pointers are not incremented so that the DSC is not lost. Attempting to read a data byte as a DSC has the same result.
2. If $\overline{DACK}$ is active the UPI-452 will attempt a DMA operation when $\overline{RD}$ or $\overline{WR}$ becomes active regardless of the DMA enable bit (MD6) in the MODE SFR. Care should be taken when using $\overline{DACK}$. For proper operation, $\overline{DACK}$ must be driven high (+5V) when not using DMA.

b. When an Input FIFO overrun error condition exists. The appropriate bits in the Host Status SFR are set and the interrupt is generated only if enabled.

OUTPUT FIFO: The Output FIFO Request for Service Interrupt operates in a similar manner as the Input FIFO interrupt:

a. When the FIFO contains the threshold number of bytes or more.

b. Output FIFO error condition interrupts are generated when the Output FIFO is underrun.

c. Data Stream Command present in the Output Buffer Latch.

A Data Stream Command interrupt is used to halt normal processing, using the command as a vector to a service routine. When DMA is disabled, the user may program (through HC1) INTRQ to include FIFO Request for Service Interrupts or use INTRQIN and INTRQOUT as Request for Service Interrupts.

IMMEDIATE COMMAND INTERRUPTS:

a. An Immediate Command Out Interrupt is generated, if enabled, to the Host and the corresponding Host Status SFR bit (HSTAT HST6) is cleared, when the internal CPU writes to the Immediate Command OUT (IMOUT) SFR. When the Host reads the Immediate Command OUT (IMOUT) SFR the corresponding bit in the Host Status (HSTAT) SFR is set. This causes the Slave Status Immediate Command OUT Status bit (SSTAT SST6) to be cleared indicating that the Immediate Command OUT (IMOUT) SFR is empty. If enabled, a FIFO-Slave Interface will also be generated to the internal CPU. (See Figure 7b, Immediate Command OUT Flowchart.)

b. An Immediate Command IN interrupt is generated, if enabled, to the Host when the internal CPU has read a byte from the Immediate Command IN (IMIN) SFR. The read operation clears the Host Status SFR Immediate Command IN Status bit (HSTAT HST2) indicating that the Immediate Command IN SFR is empty. The corresponding Slave Status (SSTAT) SFR bit is also set to indicate an empty status. Setting the Slave Status SFR bit generates a FIFO-Slave Interface interrupt, if enabled, to the internal CPU. (See Figure 7a, Immediate Command IN Flowchart.)

NOTE:
Immediate Command IN and OUT interrupts are actually specific Request For Service interrupts to the Host.

FIFO DMA FREEZE MODE: When the internal CPU invokes FIFO DMA Freeze Mode, for example at reset or to reconfigure the FIFO interface, INTRQ is activated. The INTRQ can only be deactivated by the external Host reading the Host Status SFR (HST1 remains active until FIFO DMA Freeze Mode is disabled by the internal CPU).

Once an interrupt is generated, INTRQ will remain high until no interrupt generating condition exists. For a FIFO underrun/overrun error interrupt, the interrupt condition is deactivated by the external Host reading the Host Status SFR. An interrupt is serviced by reading the Host Status SFR to determine the source of the interrupt and vectoring the appropriate service routine.

DMA Requests to the Host

The UPI-452 generates two DMA requests, DRQIN and DRQOUT, to facilitate data transfer between the Host and the Input and Output FIFO channels. A DMA acknowledge, $\overline{DACK}$, is used as a chip select and initiates a data transfer. The external $\overline{READ}$ and $\overline{WRITE}$ signals select the Input and Output FIFO respectively. The $\overline{CS}$ and address lines can also be used as a DMA acknowledge for processors with onboard DMA controllers which do not generate a $\overline{DACK}$ signal.

The internal CPU can configure the UPI-452 to request service from the external host via DMA or interrupts by programming Mode SFR MD6 bit. In addition the external Host enables DMA requests through bits 6 and 7 of the Host Control SFR. When a DMA request is invoked the number of bytes transferred to the Input FIFO is the total number of bytes in the Input FIFO (as determined by the CBP SFR) minus the value programmed in the Input FIFO Threshold SFR. The DMA request line is activated only when the Input FIFO has a threshold number of bytes that can be transferred.

The Output FIFO DMA request is activated when a DSC is written by the internal CPU at the end of a less than threshold size block of data (Flush Mode) or when the Output FIFO threshold is reached. The request remains active until the Input FIFO becomes full or the Output FIFO becomes empty. If a DSC is encountered during an Output FIFO DMA transfer, the DMA request is dropped until the DSC is read. The DMA request will be reactivated after the DSC is read and remains active until the Output FIFO becomes empty or another DSC is encountered.

FIFO MODULE - INTERNAL CPU INTERFACE

Overview

The Input and Output FIFOs are accessed by the internal CPU through direct addressing of the FIFO IN/COMMAND IN and FIFO OUT/COMMAND OUT Special Function Registers. All of the 80C51 instructions involving direct addressing may be used to access the FIFO's SFRs. The FIFO IN, COMMAND IN and Immediate Command In SFRs are actually read only registers, and their Output counterparts are write only. Internal DMA transfers data between Internal memory, External Memory and the Special Function Registers. The Special Function Registers appear as another group of dedicated memory addresses and are programmed as the source or destination via the DMA0/DMA1 Source Address or Destination Address Special Function Registers. The FIFO module manages the transfer of data between the external host and FIFO SFRs.

Internal CPU Access to FIFO Via Software Instructions

The internal CPU has access to the Input and Output FIFOs via the FIFO IN/COMMAND IN and FIFO OUT/COMMAND OUT SFRs which reside in the Special Function Register Array. At the end of every instruction that involves a read of the FIFO IN/COMMAND IN SFR, the SFR is written over by a new byte from the Input FIFO channel when available. At the end of every instruction that involves a write to the FIFO OUT/COMMAND OUT SFR, the new byte is written into the Output FIFO channel and the write pointer is incremented after the write operation (post incremented).

The internal CPU reads the Input FIFO by using the FIFO IN/COMMAND IN SFR as the source register in an instruction. Those instructions which read the Input FIFO are listed below:

```
ADD  A,FIFO IN/COMMAND IN
ADDC A,FIFO IN/COMMAND IN
PUSH FIFO IN/COMMAND IN
ANL  A,FIFO IN/COMMAND IN
ORL  A,FIFO IN/COMMAND IN
XRL  A,FIFO IN/COMMAND IN
CJNE A,FIFO IN/COMMAND IN, rel
SUBB A,FIFO IN/COMMAND IN
MOV  direct,FIFO IN/COMMAND IN
MOV  @Ri,FIFO IN/COMMAND IN
MOV  Rn,FIFO IN/COMMAND IN
MOV  A,FIFO IN/COMMAND IN
```

After each access to these registers, they are overwritten by a new byte from the FIFO.

NOTE:
Instructions which use the FIFO IN or COMMAND IN SFR as both a source and destination register will have the data destroyed as the next data byte is rewritten into the FIFO IN register at the end of the instruction. These instructions are not supported by the UPI-452 FIFO. Data can only be read through the FIFO IN SFR and DSCs through the COMMAND IN SFR. Data read through the COMMAND IN SFR will be read as 0FFH, and DSCs read through the FIFO IN SFR will be read as OFFH. The Immediate Command in SFR is read with the same instructions as the FIFO IN and COMMAND IN SFRs.

The FIFO IN, COMMAND IN and Immediate Command In SFRs are read only registers. Any write operation performed on these registers will be ignored and the FIFO pointers will remain intact.

The internal CPU uses the FIFO OUT SFR to write to the Output FIFO and any instruction which uses the FIFO OUT or COMMAND OUT SFR as a destination will invoke a FIFO write. DSCs are differentiated from data by writing to the COMMAND OUT SFR. In the FIFO, Data Stream Commands have the ninth bit associated with the command byte set to "1". The instructions used to write to the Output FIFO are listed below:

```
MOV FIFO OUT/COMMOUT, A
MOV FIFO OUT/COMMOUT, direct
MOV FIFO OUT/COMMOUT, Rn
POP FIFO OUT/COMMOUT
MOV FIFO OUT/COMMOUT, #data
MOV FIFO OUT/COMMOUNT, @Ri
```

NOTE:
Instructions which use the FIFO OUT/COMMAND OUT SFRs as both a source and destination register cause invalid data to be written into the Output FIFO. These instructions are not supported by the UPI-452 FIFO.

GENERAL PURPOSE DMA CHANNELS

Overview

There are two identical General Purpose DMA Channels on the UPI-452 which allow high speed data transfer from one writeable memory space to another. As many as 64K bytes can be transferred in a single DMA operation. The following memory spaces can be used with DMA channels:

- Internal Data Memory
- External Data Memory
- Special Function Registers

The Special Function Register array appears as a limited group of dedicated memory addresses. The Special Function Registers may be used in DMA transfer operations by specifying the SFR as the source or destination address. The Special Function Registers which may be used in DMA transfers are listed in Table 4. Table 4 also shows whether the SFR may be used as Source or Destination only, or both.

The FIFO can be accessed during DMA by using the FIFO IN SFR as the DMA Source Address Register (SAR) or the FIFO OUT SFR as the Destination Address Register (DAR). (Note: Since the FIFO IN SFR is a read only register, the DMA transfer will be ignored if it is used as a DMA DAR. This is also true if the FIFO OUT SFR is used as a DMA SAR.)

Each DMA channel is software programmable to operate in either Block Mode or Demand Mode. In the Block Mode, DMA transfers can be further programmed to take place in Burst Mode or Alternate Cycle mode. In Burst Mode, the processor halts its execution and dedicates its resources to the DMA transfer. In Alternate Cycle Mode, DMA cycles and instruction cycles occur alternately.

In Demand Mode, a DMA transfer occurs only when it is demanded. Demands can be accepted from an external device (through External Interrupt pins, EXT0/EXT1) or from either the Serial Channel or FIFO flags. In this way, a DMA transfer can be synchronized to an external device, the FIFO or the Serial Port. If the External Interrupt is configured in Edge Mode, a single byte transfer occurs per transition. The external interrupt itself will occur if enabled. If the External Interrupt is configured in Level Mode, DMA transfers continue until the External Interrupt request goes inactive or the byte count becomes zero. The following flags activate Demand Mode transfers of one byte to/from the FIFO or Serial Channel:

RI - Serial Channel Receiver Buffer Full

TI - Serial Channel Transmitter Buffer Empty

Architecture

There are three 16 bit and one 8 bit Special Function Registers associated with each DMA channel.

- The 16 bit Source Address SFR (SAR) points to the source byte.
- The 16 bit Destination Address SFR (DAR) points to the destination.
- The 16 bit Byte Count SFR (BCR) contains the number of bytes to be transferred and is decremented when a byte transfer is accomplished.
- The DMA Control SFR (DCON) is eight bits wide and specifies the source memory space, destination memory space and the mode of operation.

In Auto Increment mode, the Source Address and/or Destination Address is incremented when a byte is transferred. When a DMA transfer is complete (BCR = 0), the DONE bit is set and a maskable interrupt is generated. The GO bit must be set to start any DMA transfer (also, the Slave Control SFR FRZ bit must be set to disable FIFO DMA Freeze Mode). The two DMA channels are designated as DMA0 and DMA1, and their corresponding registers are suffixed by 0 or 1; e.g. SAR0, DAR1, etc.

Table 4. DMA Accessible Special Function Registers

SFR	Symbol	Address	Source Only	Destination Only	Either
Accumulator	A/ACC	0E0H			Y
B Register	B	0F0H			Y
FIFO IN	FIN	0EEH	Y		
COMMAND IN	CIN	0EFH	Y		
FIFO OUT	FOUT	0FEH		Y	
COMMAND OUT	COUT	0FFH		Y	
Serial Data Buffer	SBUF	099H			Y
Port 0	P0	080H			Y
Port 1	P1	090H			Y
Port 2	P2	0A0H			Y
Port 3	P3	0B0H			Y
Port 4	P4	0C0H			Y

DMA Special Function Registers

DMA Control SFR: DCON0, DCON1

Symbolic Address									Physical Address
DCON0	DAS	IDA	SAS	ISA	DM	TM	DONE	GO	092H
DCON1	DAS	IDA	SAS	ISA	DM	TM	DONE	GO	093H
	(MSB)							(LSB)	

Reset Status: DCON0 and DCON1 = 00H

Bit Definition:

DAS	IDA	Destination Address Space
0	0	External Data Memory without Auto-Increment
0	1	External Data Memory with Auto-Increment
1	0	Special Function Register
1	1	Internal Data Memory

SAS	ISA	Source Address Space
0	0	External Data Memory without Auto-Increment
0	1	External Data Memory with Auto-Increment
1	0	Special Function Register
1	1	Internal Data Memory

DM	TM	DMA Transfer Mode
0	0	Alternate-Cycle Transfer Mode
0	1	Burst Transfer Mode
1	0	FIFO or Serial Channel Demand Mode
1	1	External Demand Mode

DONE DMA transfer Flag:

0 DMA transfer is not completed.

1 DMA transfer is complete.

NOTE:
This flag is set when contents of the Byte Count SFR decrements to zero. It is reset automatically when the DMA vectors to its interrupt routine.

GO Enable DMA Transfer:

0 Disable DMA transfer (in all modes).

1 Enable DMA transfer. If the DMA is in the Block mode, start DMA transfer if possible. If it is in the Demand mode, enable the channel and wait for a demand.

NOTE:
The GO bit is reset when the BCR decrements to zero.

DMA Transfer Modes

The following four modes of DMA operation are possible in the UPI-452.

1. ALTERNATE-CYCLE MODE

General

Alternate cycle mode is useful when CPU processing must occur during the DMA transfers. In this mode, a DMA cycle and an instruction cycle occur alternately. The interrupt request is generated (if enabled) at the end of the process, i.e. when BCR decrements to zero. The transfer is initiated by setting the GO bit in the DCON SFR.

Alternate-Cycle FIFO Demand Mode

Alternate cycle demand mode is useful for FIFO transfers of a less urgent nature. As mentioned before, CPU instruction cycles are interleaved with DMA transfer cycles, allowing true parallel processing.

This mode differs from FIFO Demand Mode in that CPU instruction cycles must be interleaved with DMA transfers, even if the FIFO is demanding DMA. In FIFO Demand Mode, CPU cycles would never occur if the FIFO demand was present.

Input Channel

The DMA is configured as in FIFO Demand Mode and transfers are initiated whenever an Input FIFO service request is generated. DMA transfer cycles are alternated with instruction execution cycles. DMA transfers are terminated as in FIFO Demand Mode.

Output Channel

The DMA is configured as in FIFO Demand Mode and transfers are initiated whenever an Output FIFO requests service. DMA transfer cycles are alternated with instruction execution cycles. DMA transfers are terminated as in FIFO Demand Mode.

The FIFO logic resets the interrupt flag after transferring the byte, so the interrupt is never generated.

Once the DMA is programmed to service the FIFO, the request for service interrupt for the FIFO is inhibited until the DMA is done (BCR = 0).

2. BURST MODE

In BURST mode the DMA is initiated by setting the GO bit in the DCON SFR. The DMA operation continues until BCR decrements to zero (zero byte count), then an interrupt is generated (if enabled). No interrupts are recognized during a DMA operation once started.

Input Channel

The FIFO Input Channel can be used in burst mode by specifying the FIFO IN SFR as the DMA Source Address. DMA transfers begin when the GO bit in the DMA Control SFR is set. The number of bytes to be transferred must be specified in the Byte Count SFR (BCR) and auto-incrementing of the SAR must be disabled. Once the GO bit is set nothing can interrupt the transfer of data until the BCR is zero. In this mode, a Data Stream Command encountered in the FIFO will be held in the COMMAND IN SFR with the pointers frozen, and invalid data (FFH) will be read through the FIFO IN SFR. If the input FIFO becomes empty during the block transfer, an 0FFH will be read until BCR decrements to zero.

Output Channel

The Output FIFO Channel can be used in burst mode by specifying the FIFO OUT or COMMAND OUT SFR as the DMA Destination Address. DMA transfers begin when the GO bit is set. This mode can be used to send a block of data or a block of Data Stream Commands. If the FIFO becomes full during the block transfer, the remaining data will be lost.

NOTE:
All interrupts including FIFO interrupts are not recognized in Burst Mode. Burst Mode transfers should be used to service the FIFO only when the user is certain that no Data Stream Commands are in the block to be transferred (Input FIFO) and that the FIFO contains enough space to store the block to be transferred. In all other cases Alternate Cycle or Demand Mode should be used.

3. FIFO AND SERIAL CHANNEL DEMAND MODES

NOTES:
1. If the output FIFO is configured as a one byte buffer and the user program consists of two-cycle instructions only, then Alternate-Cycle Mode should be used.
2. In non-auto increment mode for internal to external, or external to internal transfers, the lower 8 bits of the external address should not correspond to the FIFO or Serial Port address.

FIFO Demand Mode

Although any DMA mode is possible using the FIFO buffer, only FIFO Demand and Alternate Cycle FIFO Demand Modes are recommended. FIFO Demand Mode DMA transfers using the input FIFO Channel are set-up by setting the GO bit and specifying the FIFO IN register as the DMA Source Address Register. The BCR should be set to the maximum number of expected transfers. The user must also program bit 1 of the Slave Control Register (SC1) to determine whether the Slave Status (SSTAT) SFR FIFO Request For Service Flag will be activated when the FIFO becomes not empty or full. Once the Request For Service Flag is activated by the FIFO, the DMA transfer begins, and continues until the request flag is deactivated. While the request is active, nothing can interrupt the DMA (i.e. it behaves like burst mode). The DMA Request is held active until one of the following occurs:

1) The FIFO becomes empty.
2) A Data Stream Command is encountered (this generates a FIFO interrupt and DMA operation resumes after the Data Stream Command is read).
3) BCR = 0 (this generates a DMA interrupt and sets the DONE bit).

DMA transfers to the Output FIFO Channel are similar. The FIFO OUT or COMMAND OUT SFR is the DMA Destination Address SFR and a transfer is started by setting the GO bit. The user programs bit 0 of the Slave Control SFR (SC0) to determine whether a demand occurs when the Output FIFO is not full or empty. DMA transfers begin when the Request For Service Flag is activated by the FIFO logic and continue as long as the flag is active. The Flag remains active until one of the following occurs:

1) The FIFO becomes full
2) BCR = 0 (this generates a DMA interrupt and sets the DONE bit).

As in Alternate Cycle FIFO Demand Mode, the FIFO logic resets the interrupt flag after transferring the byte, so the interrupt is never generated.

After the GO bit is set, the DMA is activated if one of the following conditions takes place:

SAR(0/1) = FIFO IN and HIFRS flag is set
DAR(0/1) = FIFO OUT and HOFRS flag is set

The HIFRS and HOFRS signals are internal flags which are not accessible by software. These flags are similar to the SST0 and SST4 flags in the Slave Status Register except that they are of the opposite polarity and once set they are not cleared until the Input FIFO becomes empty (HIFRS) or the Output FIFO becomes full (HOFRS).

Serial Channel Demand Mode

Serial Channel Demand Mode is the logical choice when using the Serial Port. The DMAs can be activated by one of the Serial Channel Flags. Receiver interrupt (RI) or Transmitter Interrupt (TI).

SAR(0/1) = SBUF and RI flag is set
DAR(0/1) = SBUF and TI flag is set

NOTE:
TI flag must be set by software to initiate the first transfer.

When the DMA transfer begins, only one byte is transferred at a time. The serial port hardware automatically resets the flag after completion of the transfer, so an interrupt will not be generated unless DMA servicing is held off due to the DMA being done (BCR = 0) or when the Hold/Hold Acknowledge logic is used and the DMA does not own the bus. In this case a Serial Port interrupt may be generated if enabled because of the status of the RI or TI flags.

In FIFO demand mode, Alternate cycle FIFO demand mode or Serial Port demand mode only one of the following registers (SBUF, FIN or FOUT) should be used as either the SAR or DAR registers to prevent undesired transfers. For example if SAR0 = FIN and DAR0 = SBUF in demand mode, the DMA transfer will start if either the HIFRS or TI flags are set.

4. EXTERNAL DEMAND MODE

The DMA can be initiated by an external device via External interrupt 0 and 1 (INT0/INT1) pins. The INT0 pin demands DMA0 (Channel 0) and INT1 demands DMA1 (Channel 1). If the interrupts are configured in edge mode, a single byte transfer is accomplished for every request. Interrupts also result (INT0 and INT1) after every byte transfer (if enabled). If the interrupts are configured in level mode, the DMA transfer continues until the request goes inactive or BCR = 0. In either case, a DMA interrupt is generated (if enabled) when BCR = 0. The GO bit must be set for the transfer to begin.

EXTERNAL MEMORY DMA

When transferring data to or from external memory via DMA, the HOLD (HLD) and HOLD-ACKNOWLEDGE (HLDA) signals are used for handshaking. The HOLD and HOLD-ACKNOWLEDGE are active low signals which arbitrate control of the local bus. The UPI-452 can be used in a system where multimasters are connected to a single parallel Address/Data bus. The HLD/HLDA signals are used to share resources (memory, peripherals, etc.) among all the processors on the local bus. The UPI-452 can be configured in any of three different External Memory Modes controlled by bits 5 and 6 (REQ & ARB) in the PCON SFR (Table 5). Each mode is described below:

REQUESTER MODE: In this mode, the UPI-452 is not the bus master, but must request the bus from another device. The UPI-452 configures port pin P1.5 as a HLD output and pin P1.6 as a HLDA input. The UPI-452 issues a HLD signal when it needs external access for a DMA channel. It uses the local bus after receiving the HLDA signal from the bus master, and will not release the bus until its DMA operation is complete.

ARBITER MODE: In this mode, the UPI-452 is the bus master. It configures port pin P1.5 as HLD input and pin P1.6 as HLDA output. When a device asserts the HLD signal to use the local bus, the UPI-452 asserts the HLDA signal after current instruction execution is complete. If the UPI-452 needs an external access via a DMA channel, it waits until the requester releases the bus, HLD goes inactive.

DISABLE MODE: When external program memory is accessed by an instruction or by program counter overflow beyond the internal ROM address or external data memory is accessed by MOVX instructions, it is a local memory access and the HLD/HLDA logic is not initiated. When a DMA channel attempts data transfer to/from the external data memory, the HLD/HLDA logic is initiated as described below. DMA transfers from the internal memory space to the internal memory space does not initiate the HLD/HLDA logic.

The balance of the PCON SFR bits are described in the "80C51 Register Description: Power Control SFR" section below.

Latency

When the GO bit is set, the UPI-452 finishes the current instruction before starting the DMA operation. Thus the maximum latency is 3.0 microseconds (at 16 MHz).

DMA Interrupt Vectors

Each DMA channel has a unique vectored interrupt associated with it. There are two vectored interrupts associated with the two DMA channels. The DMA interrupts are enabled and priorities set via the Interrupt Enable and Priority SFR (see "Interrupts" section). The interrupt priority scheme is similar to the scheme in 80C51.

Table 5. DMA MODE CONTROL - PCON SFR

Symbolic Address									Physical Address
PCON	—*	ARB	REQ	—*	—*	—*	—*	—*	87H
	(MSB)							(LSB)	

*Defined as per MLS-51 Data Sheet
Reset Status: 00H

Definition:

ARB	REQ	
0	0	HLD/HLDA logic is disabled.
0	1	The UPI-452 is in the Requester Mode.
1	0	The UPI-452 is in the Arbiter Mode.
1	1	Invalid

When a DMA operation is complete (BCR decrements to zero), the DONE flag in the respective DCON (DCON0 or DCON1) SFR is set. If the DMA interrupt is enabled, the DONE flag is reset automatically upon vectoring to the interrupt routine.

Interrupts When DMA is Active

If a Burst Mode DMA transfer is in progress, the interrupts are not serviced until the DMA transfer is complete. This is also true for level activated External Demand DMA transfers. During Alternate Cycle DMA transfers, however, the interrupts are serviced at the end of the DMA cycle. After that, DMA cycles and instruction execution cycles occur alternately. In the case of edge activated External Demand Mode DMA transfers, the interrupt is serviced at the end of DMA transfer of that single byte.

DMA Arbitration

Only one of the two DMA channels is active at a time, except when both are configured in the Alternate Cycle mode. In this case, the DMA cycles and Instruction Execution cycles occur in the following order:

1. DMA Cycle 0.
2. Instruction execution.
3. DMA Cycle 1.
4. Instruction execution.

DMA0 has priority over DMA1 during simultaneous activation of the two DMA channels. If one DMA channel is active, the other DMA channel, if activated, waits until the first one is complete.

If DMA0 is already in the Alternate Cycle mode and DMA1 is activated in Alternate Cycle Mode, it will take two instruction cycles before DMA1 is activated (due to the priority of DMA0). Once DMA1 becomes active, the execution will follow the normal sequence.

If DMA0 is already in the Alternate Cycle mode and DMA1 is activated in Burst Mode, the DMA1 Burst transfer will follow the DMA0 Alternate Cycle transfer (after the completion of the next instruction).

If the UPI-452 (as a Requester) asserts a HLD signal to request a DMA transfer (see "External Memory DMA")and its other DMA Channel requests a transfer before the HLDA signal is received, the channel having higher priority is activated first. A Burst Mode transfer on channel 0 can not be interrupted since DMA0 has the highest priority. A Demand Mode transfer on channel 0 is the only type of activity that can interrupt a block transfer on DMA1.

If, while executing a DMA transfer, the Arbiter receives a HLD signal, and then before it can acknowledge, its other DMA Channel requests a transfer, it then completes the second DMA transfer before sending the HLDA signal to release the bus to the HLD request.

DMA transfers may be held off under the following conditions:

1. A write to any of the DMA registers inhibits the DMA for one instruction cycle.

NOTE:
An instruction cycle may be executed in 1, 2 or 4 machine cycles dependent on the instruction being executed. DMA transfers are only executed after the completion of an instruction cycle never between machine cycles of a single instruction cycle. Similarly instruction cycles are only executed upon completion of a DMA transfer whether it be a one machine cycle transfer or two machine cycles (for ext. to ext. memory transfers).

2. A single machine cycle DMA register read operation (i.e. MOV A, DCON0) will inhibit the DMA for one instruction cycle. However a two cycle DMA register read operation will not inhibit the DMA (i.e. MOV P1, DCON0).

If the HOLD/HOLD Acknowledge logic is enabled in requestor mode the hold request will go active once the go bit has been set (for burst mode) and once the demand flag is set (for demand mode) regardless of whether the DMA is held off by one of the above conditions.

The DMA Transfer waveforms are in Figures 8-11.

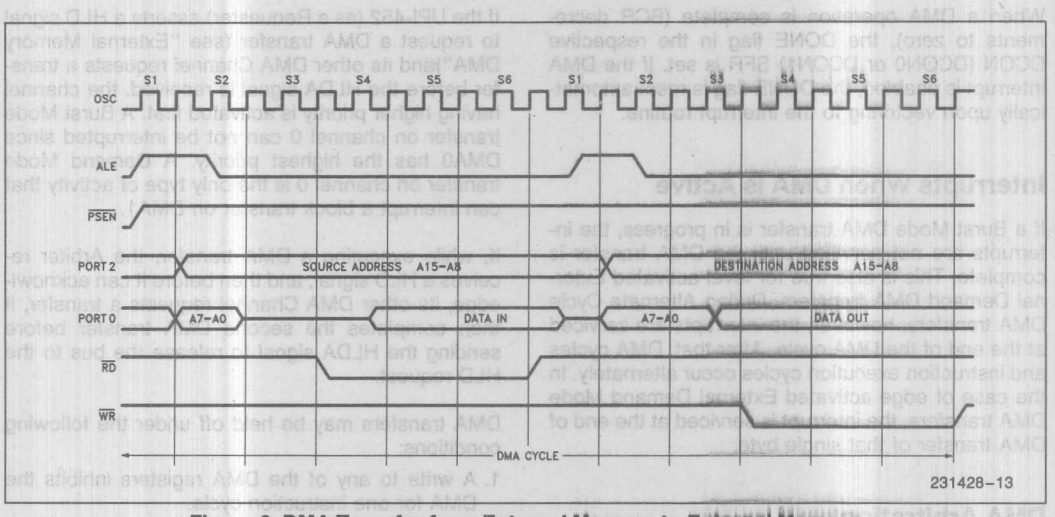


Figure 8. DMA Transfer from External Memory to External Memory

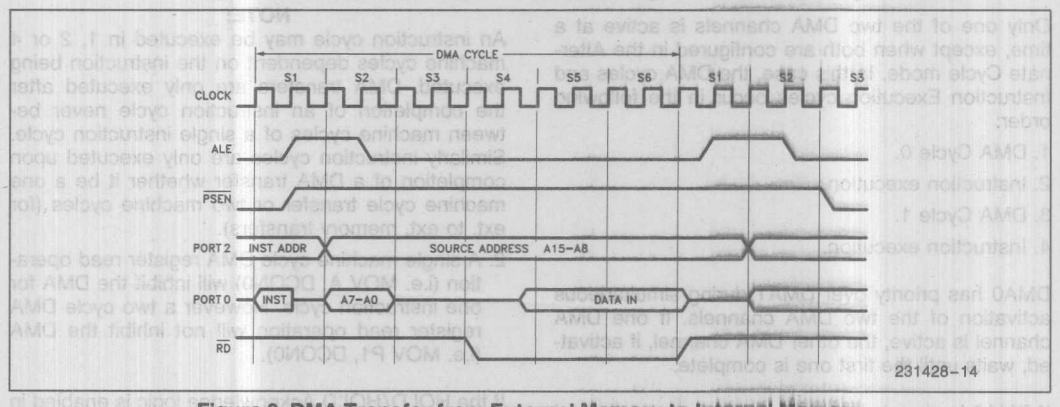


Figure 9. DMA Transfer from External Memory to Internal Memory

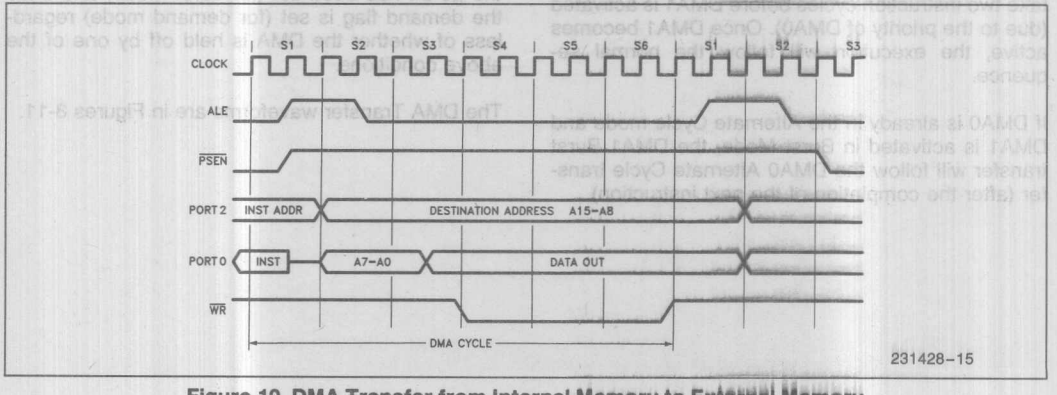


Figure 10. DMA Transfer from Internal Memory to External Memory

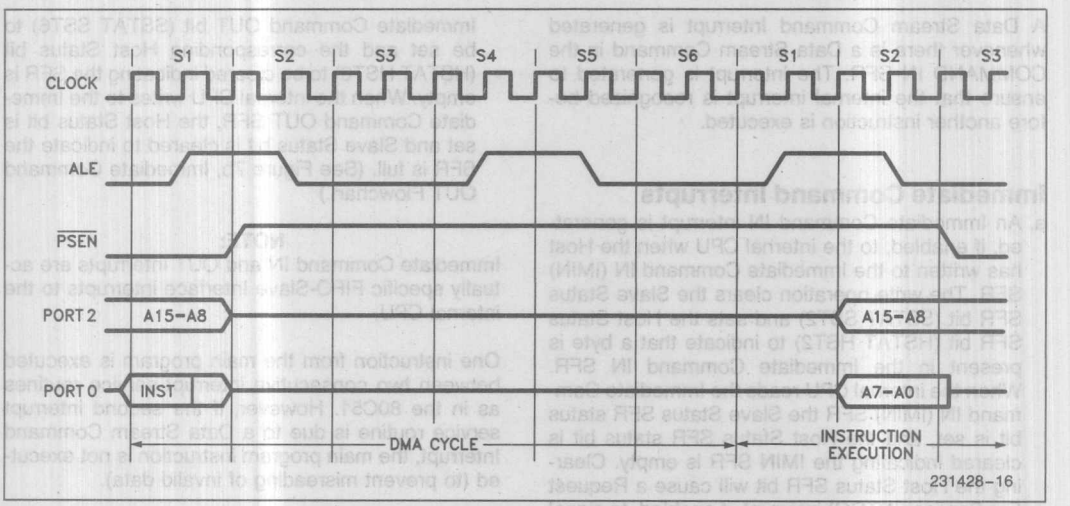


Figure 11. DMA Transfer from Internal Memory to Internal Memory

INTERNAL INTERRUPTS

Overview

The UPI-452 provides a total of eight interrupt sources (Table 6). Their operation is the same as in the 80C51, with the addition of three new interrupt sources for the UPI-452 FIFO and DMA features. These added interrupts have their enable and priority bits in the Interrupt Enable and Priority (IEP) SFR. The IEP SFR is in addition to the 80C51 Interrupt Enable (IE) and Interrupt Priority (IP) SFRs. The added interrupt sources are also globally enabled or disabled by the EA bit in the Interrupt Enable SFR. Table 6 lists the eight interrupt sources in order of priority. Table 7 lists the eight interrupt sources and their respective address vector location in program memory. (DMA interrupts are discussed in the "General Purpose DMA Channels" section. Additional interrupt information for Timer/Counter, Serial Channel, External Interrupt may be found in the Microcontroller Handbook for the 80C51.)

Table 6. Interrupt Priority

Interrupt Source	Priority Level
	(highest)
External Interrupt 0	0
Internal Timer/Counter 0	1
DMA Channel 0 Request	2
External Interrupt 1	3
DMA Channel 1 Request	4
Internal Timer/Counter 1	5
FIFO - Slave Bus Interface	6
Serial Channel	7
	(lowest)

Table 7. Interrupt Vector Addresses

Interrupt Source	Starting Address
External Interrupt 0	3 (003H)
Internal Timer/Counter 0	11 (00BH)
External Interrupt 1	19 (013H)
Internal Timer/Counter 1	27 (01BH)
Serial Channel	35 (023H)
FIFO - Slave Bus Interface	43 (02BH)
DMA Channel 0 Request	51 (033H)
DMA Channel 1 Request	59 (03BH)

FIFO Module Interrupts to Internal CPU

The FIFO module generates interrupts to the internal CPU whenever the FIFO requests service or when a Data Stream Command is in the COMMAND IN SFR. The Input FIFO will request service whenever it becomes full or not empty depending on bit 1 of the Slave Control SFR (IFRS). Similarly, the Output FIFO requests service when it becomes empty or not full as determined by bit 0 of the Slave Control SFR (OFRS). Request for Service interrupts are generated only if enabled by the internal CPU via the Interrupt Enable SFR, and the Slave Control Register.

A Data Stream Command Interrupt is generated whenever there is a Data Stream Command in the COMMAND IN SFR. The interrupt is generated to ensure that the internal interrupt is recognized before another instruction is executed.

Immediate Command Interrupts

a. An Immediate Command IN interrupt is generated, if enabled, to the internal CPU when the Host has written to the Immediate Command IN (IMIN) SFR. The write operation clears the Slave Status SFR bit (SSTAT SST2) and sets the Host Status SFR bit (HSTAT HST2) to indicate that a byte is present in the Immediate Command IN SFR. When the internal CPU reads the Immediate Command IN (IMIN) SFR the Slave Status SFR status bit is set, and the Host Status SFR status bit is cleared indicating the IMIN SFR is empty. Clearing the Host Status SFR bit will cause a Request For Service (INTRQ) interrupt, if enabled, to signal the Host that the IMIN SFR is empty. (See Figure 7a, Immediate Command IN Flowchart.)

b. An Immediate Command OUT interrupt is generated, if enabled, to the internal CPU when the Host has read the Immediate Command OUT SFR. The Host read causes the Slave Status Immediate Command OUT bit (SSTAT SST6) to be set and the corresponding Host Status bit (HSTAT HST6) to be cleared indicating the SFR is empty. When the internal CPU writes to the Immediate Command OUT SFR, the Host Status bit is set and Slave Status bit is cleared to indicate the SFR is full. (See Figure 7b, Immediate Command OUT Flowchart.)

NOTE:
Immediate Command IN and OUT interrupts are actually specific FIFO-Slave Interface interrupts to the internal CPU.

One instruction from the main program is executed between two consecutive interrupt service routines as in the 80C51. However, if the second interrupt service routine is due to a Data Stream Command Interrupt, the main program instruction is not executed (to prevent misreading of invalid data).

Interrupt Enabling and Priority

Each of the three interrupt special function registers (IE, IP and IEP) is listed below with its corresponding bit definitions.

Interrupt Enable SFR (IE)

Symbolic Address									Physical Address
IE	EA	—	—	ES	ET1	EX1	ET0	EX0	0A8H
	(MSB)							(LSB)	

Symbol	Position	Function
EA	IE.7	Enables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	(reserved)
—	IE.5	(reserved)
ES	IE.4	Serial Channel interrupt enable
ET1	IE.3	Internal Timer/Counter 1 Overflow Interrupt
EX1	IE.2	External Interrupt Request 1.
ET0	IE.1	Internal Timer/Counter 0 Overflow Interrupt
EX0	IE.0	External Interrupt Request 0.

Interrupt Priority SFR (IP)

A priority level of 0 or 1 may be assigned to each interrupt source, with 1 being higher priority level, through the IP and the IEP (Interrupt Enable and Priority) SFR. A priority level of 1 interrupt can interrupt a priority level 0 service routine to allow nesting of interrupts.

Symbolic Address									Physical Address
IP	—	—	—	PS	PT1	PX1	PT0	PX0	0B8H
	(MSB)							(LSB)	

Symbol	Position	Function	Priority Within A Level
			(lowest)
—	IP.7	(reserved)	—
—	IP.6	(reserved)	—
—	IP.5	(reserved)	—
PS	IP.4	Local Serial Channel	0.7
PT1	IP.3	Internal Timer/Counter 1	0.5
PX1	IP.2	External Interrupt Request 1	0.3
PT0	IP.1	Internal Timer/Counter 0	0.1
PX0	IP.0	External Interrupt Request 0	0.0
			(highest)

Interrupt Enable and Priority SFR (IEP)

The Interrupt Enable and Priority Register establishes the enabling and priority of those resources not covered in the Interrupt Enable and Interrupt Priority SFRs.

Symbolic Address									Physical Address
IEP	—	—	PFIFO	EDMA0	EDMA1	PDMA0	PDMA1	EFIFO	0F8H
	(MSB)							(LSB)	

Symbol	Position	Function	Priority Within a Level
—	IEP.7	(reserved)	
—	IEP.6	(reserved)	
PFIFO	IEP.5	FIFO Slave Bus Interface Interrupt Priority	0.6
EDMA0	IEP.4	DMA Channel 0 Interrupt Enable	
EDMA1	IEP.3	DMA Channel 1 Interrupt Enable	
PDMA0	IEP.2	DMA Channel 0 Priority	0.2
PDMA1	IEP.1	DMA Channel 1 Priority	0.4
EFIFO	IEP.0	FIFO Slave Bus Interface Interrupt Enable	

FIFO-EXTERNAL HOST INTERFACE FIFO DMA FREEZE MODE

Overview

During FIFO DMA Freeze Mode the internal CPU can reconfigure the FIFO interface. FIFO DMA Freeze Mode is provided to prevent the Host from accessing the FIFO during a reconfiguration sequence. The internal CPU invokes FIFO DMA Freeze Mode by clearing bit 3 of the Slave Control SFR (SC3). INTRQ becomes active whenever FIFO DMA Freeze Mode is invoked to indicate the freeze status. The interrupt can only be deactivated by the Host reading the Host Status SFR.

During FIFO DMA Freeze Mode only two operations are possible by the Host to the UPI-452 slave, the balance are disabled, as shown in Table 8. The internal DMA is disabled during FIFO DMA Freeze Mode, and the internal CPU has write access to all of the FIFO control SFRs (Table 9).

Initialization

At power on reset the FIFO Host interface is automatically frozen. The Slave Control Enable FIFO DMA Freeze Mode bit defaults to FIFO DMA Freeze Mode (SLCON FRZ = 0). Below is a list of the FIFO Special Function Registers and their default power on reset values;

SFR Name	Label	Value
Channel Boundary Pointer	CBP	40H / 64D
Output Channel Read Pointers	ORPR	40H / 64D
Output Channel Write Pointers	OWPR	40H / 64D
Input Channel Read Pointers	IRPR	00H / 00D
Input Channel Write Pointers	IWPR	00H / 00D
Input Threshold	ITHR	80H / 128D
Output Threshold	OTHR	01H / 1D

The Input and Output FIFO channels may be reconfigured by programming any of these SFRs while the FIFO Host interface is in FIFO DMA Freeze Mode. The UPI-452 also notifies the Host that FIFO DMA Freeze Mode is in progress by setting the Host Status SFR FIFO DMA Freeze Mode Status bit, FIFO DMA Freeze Mode In Progress. The Host interrogates the Host Status SFR to determine the status of the FIFO Host interface following reset before attempting to read from or write to the UPI-452 FIFO buffer.

Table 8. Slave Bus Interface Status During FIFO DMA Freeze Mode

Interface Pins; $\overline{DACK}$	$\overline{CS}$	A2	A1	A0	$\overline{READ}$	$\overline{WRITE}$	Operation In Normal Mode	Status In FIFO DMA Freeze Mode
1	0	0	1	0	0	1	Read Host Status SFR	Operational
1	0	0	1	1	0	1	Read Host Control SFR	Operational
1	0	0	1	1	1	0	Write Host Control SFR	Disabled
1	0	0	0	0	0	1	Data or DMA Data from Output Channel	Disabled
1	0	0	0	0	1	0	Data or DMA Data to Input Channel	Disabled
1	0	0	0	1	0	1	Data Stream Command from Output Channel	Disabled
1	0	0	0	1	1	0	Data Stream Command to Input Channel	Disabled
1	0	1	0	0	0	1	Read Immediate Command Out from Output Channel	Disabled
1	0	1	0	0	1	0	Write Immediate Command In to Input Channel	Disabled
0	X	X	X	X	0	1	DMA Data from Output Channel	Disabled
0	X	X	X	X	1	0	DMA Data to Input Channel	Disabled

The UPI-452 can also be programmed to interrupt the Host following power on reset in order to indicate to the Host that FIFO DMA Freeze Mode is in progress. This is done by enabling the INTRQ interrupt output pin via the MODE SFR (MD4) before the Slave Control SFR Enable FIFO DMA Freeze Mode bit is set to Normal Mode. At power on reset the Mode SFR is forced to zero. This disables all interrupt and DMA output pins (INTRQ, DRQIN/INTRQIN and DRQOUT/INTRQOUT). Because the Host Status SFR FIFO DMA Freeze Mode In Progress bit is set, a Request For Service, INTRQ, interrupt is pending until the Host Status SFR is read. This is because the FIFO DMA Freeze Mode interrupt is always enabled. If the Slave Control FIFO DMA Freeze Mode bit (SLCON FRZ) is set to Normal Mode before the MODE SFR INTRQ bit is enabled, the INTRQ output will not go active when the MODE SFR INTRQ bit is enabled if the Host Status SFR has been read.

The default values for the FIFO and Slave Interface represents minimum UPI-452 internal initialization. No specific Special Function Register initialization is required to begin operation of the FIFO Slave Interface. The last initialization instruction must always set the UPI-452 to Normal Mode. This causes the UPI-452 to exit FIFO DMA Freeze Mode and enables Host read/write access of the FIFO.

Following reset, either hardware (via the RST pin) or software (via HCON SFR bit HC3) the UPI-452 requires 2 internal machine cycles (24 TCLCL) to update all internal registers.

Invoking FIFO DMA Freeze Mode During Normal Operation

When the UPI-452 is in normal operation, FIFO DMA Freeze Mode should not be arbitrarily invoked by clearing SC3 (SC3 = 0) because the external Host runs asynchronously to the internal CPU. Invoking FIFO DMA Freeze Mode without first stopping the external Host from accessing the UPI-452 will not guarantee a clean break with the external Host.

The proper way to invoke FIFO DMA Freeze Mode is by issuing an Immediate Command to the external host indicating that FIFO DMA Freeze Mode will be invoked. Upon receiving the Immediate Command, the external Host should complete servicing all pending interrupts and DMA requests, then send an Immediate Command back to the UPI-452 acknowledging the FIFO DMA Freeze Mode request. After issuing the first Immediate Command, the internal CPU should not perform any action on the FIFO until FIFO DMA Freeze Mode is invoked.

If FIFO DMA Freeze Mode is invoked without stopping the Host during Host transfers, only the last two bytes of data written into or read from the FIFO will be valid. The timing diagram for disabling the FIFO module to the external Host interface is illustrated in Figure 12. Due to this synchronization sequence, the UPI-452 might not go into FIFO DMA Freeze Mode immediately after SC3 is cleared. A special bit in the Slave Status Register (SST5) is provided to indicate the status of the FIFO DMA Freeze Mode. The FIFO DMA Freeze Mode operations described in this section are only valid after SST5 is cleared.

As FIFO DMA Freeze Mode is invoked, the DRQIN or DRQOUT will be deactivated (stopping the transferring of data), bit 1 of the Host Status SFR will be set (HST1 = 1), and SST5 will be cleared (SST5 = 0) to indicate to the external Host and internal CPU that the slave interface has been frozen. After the freeze becomes effective, any attempt by the external Host to access the FIFO will cause the overrun and underrun bits to be activated (bits HST7 (for reads) or HST3 (for writes)). These two bits, HST3 and HST7, will be set (deactivated) after the Host Status SFR has been read. If INTRQ is used to request service, the FIFO interface is frozen upon completion of any Host read or write operation in progress.

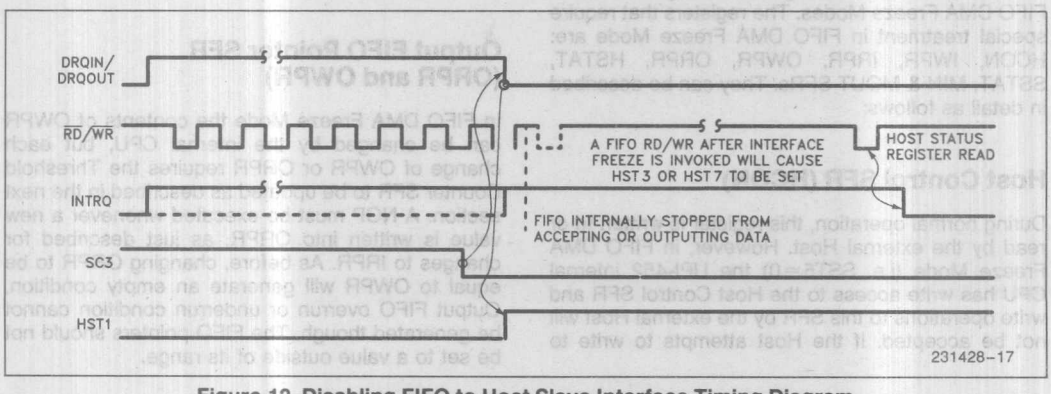


Figure 12. Disabling FIFO to Host Slave Interface Timing Diagram

External Host writing to the Immediate Command In SFR and the Host Control SFR is also inhibited when the slave bus interface is frozen. Writing to these two registers after FIFO DMA Freeze Mode is invoked will also cause HST3 (overrun) to be activated (HST3 = 0). Similarly, reading the Immediate Command Out Register by the external Host is disabled during FIFO DMA Freeze Mode, and any attempt to do so will cause the clearing (deactivating, "0") of HST7 bit (underrun).

After the slave bus interface is frozen, the internal CPU can perform the following operations on the FIFO Special Function Registers (these operations are allowed only during FIFO DMA Freeze Mode).

For FIFO Reconfiguration
1. Changing the Channel Boundary Pointer SFR.
2. Changing the Input and Output Threshold SFR.

To Enhance the Testability
3. Writing to the read and write pointers of the Input and Output FIFO's.
4. Writing to and reading the Host Control SFRs.
5. Controlling some bits of Host and Slave Status SFRS.
6. Reading the Immediate Command Out SFR and Writing to the Immediate Comand In SFR.

Description of each of these special functions are as follows:

FIFO Module SFRs During FIFO DMA Freeze Mode

Table 9 summarizes the characteristics of all the FIFO Special Function Registers during normal and FIFO DMA Freeze Modes. The registers that require special treatment in FIFO DMA Freeze Mode are: HCON, IWPR, IRPR, OWPR, ORPR, HSTAT, SSTAT, MIN & MOUT SFRs. They can be described in detail as follows:

Host Control SFR (HCON)

During normal operation, this register is written to or read by the external Host. However, in FIFO DMA Freeze Mode (i.e. SST5 = 0) the UPI-452 internal CPU has write access to the Host Control SFR and write operations to this SFR by the external Host will not be accepted. If the Host attempts to write to HCON, the Input Channel error condition flag (HST3) will be cleared.

Input FIFO Pointer Registers (IRPR & IWPR)

Once the FIFO module is in FIFO DMA Freeze Mode, error flags due to overrun and underrun of the Input FIFO pointers will be disabled. Any attempt to create an overrun or underrun condition by changing the Input FIFO pointers would result in an inconsistency in performance between the status flag and the threshold counter.

To enhance the speed of the UPI-452, read operations on the Input FIFO will look ahead by two bytes. Hence, every time the IRPR is changed during FIFO DMA Freeze Mode, two NOPs need to be executed so that the two byte pipeline can be updated with the new data bytes pointed to by the new IRPR. The Threshold Counter SFR also needs to change by the same number of bytes as the IRPR (increase Threshold Counter if IRPR goes forward or decrease if IRPR goes backward). This will ensure that future interrupts will still be generated only after a threshold number of bytes are available. (See "Input and Output FIFO Threshold SFR" section below.)

In FIFO DMA Freeze Mode, the internal CPU can also change the content of IWPR, and each change of IWPR also requires an update of the Threshold Counter SFR.

Normally, the internal CPU cannot write into the Input FIFO. It can, however, during FIFO DMA Freeze Mode by first reconfiguring the FIFO as an Output FIFO (Refer to "Input and Output FIFO Threshold SFR" section below). Changing the IRPR to be equal to IWPR generates an empty condition while changing IWPR to be equal to IRPR generates a full condition. The order in which the pointers are written determines whether a full or empty condition is generated.

Output FIFO Pointer SFR (ORPR and OWPR)

In FIFO DMA Freeze Mode the contents of OWPR can be changed by the internal CPU, but each change of OWPR or ORPR requires the Threshold Counter SFR to be updated as described in the next section. A NOP must be executed whenever a new value is written into ORPR, as just described for changes to IRPR. As before, changing ORPR to be equal to OWPR will generate an empty condition, Output FIFO overrun or underrun condition cannot be generated though. The FIFO pointers should not be set to a value outside of its range.

Table 9. FIFO SFR's Characteristics During FIFO DMA Freeze Mode

Label	Name	Normal Operation (SST5 = 1)	FIFO DMA Freeze Mode Operation (SST5 = 0)
HCON	Host Control	Not Accessible	Read & Write
HSTAT	Host Status	Read Only	Read & Write 4
SLCON	Slave Control	Read & Write	Read & Write
SSTAT	Slave Status	Read Only	Read & Write 4
IEP	Interrupt Enable & Priority	Read & Write	Read & Write
MODE	Mode Register	Read & Write	Read & Write
IWPR	Input FIFO Write Pointer	Read Only	Read & Write 5
IRPR	Input FIFO Read Pointer	Read Only	Read & Write 1, 5
OWPR	Output FIFO Write Pointer	Read Only	Read & Write 6
ORPR	Output FIFO Read Pointer	Read Only	Read & Write 2, 6
CBP	Channel Boundary Pointer	Read Only	Read & Write 3
IMIN	Immediate Command In	Read Only	Read & Write
IMOUT	Immediate Command Out	Read & Write	Read & Write
FIN	FIFO IN	Read Only	Read Only
CIN	COMMAND IN	Read Only	Read Only
FOUT	FIFO OUT	Read & Write	Read & Write
COUT	COMMAND OUT	Read & Write	Read & Write
ITHR	Input FIFO Threshold	Read Only	Read & Write
OTHR	Output FIFO Threshold	Read Only	Read & Write

NOTES:
1. Writing of IRPR will automatically cause the FIFO IN SFR to load the contents of the Input FIFO from that location.
2. Writing to ORPR will automatically cause the IOBL SFR to load the contents of the Output FIFO at that ORPR address.
3. Writing to the CBP SFR will cause automatic reset of the four pointers of the Input and Output FIFO channels.
4. The internal CPU cannot directly change the status of these registers. However, by changing the status of the FIFO channels, the internal CPU can indirectly change the contents of the status registers.
5. Changing the Input FIFO Read/Write Pointers also requires that a consistent update of the Input FIFO Threshold Counter SFR.
6. Changing the Output FIFO Read/Write Pointers also requires that a consistent update of the Output FIFO Threshold Counter SFR.

Input and Output FIFO Threshold SFR (ITHR & OTHR)

The Input and Output FIFO Threshold SFRs are also programmable by the internal CPU during FIFO DMA Freeze Mode. For proper operation of the Threshold feature, the Threshold SFR should be changed only when the Input and Output FIFO channels are empty, since they reflect the current number of bytes available to read/write before an interrupt is generated.

Table 10 illustrates the Threshold SFRs range of values and the number of bytes to be transferred when the Request For Service Flag is activated:

Table 10. Threshold SFRs Range of Values and Number of Bytes to be Transferred

ITHR (lower seven bits)	No. of Bytes Available to be Written	OTHR (lower seven bits	No. of Bytes Available to be Read
0	CBP	1	2
1	CBP-1	2	3
2	CBP-2	3	4
•	•	•	•
•	•	•	•
•	•	•	•
CBP-3	3	(80H-CBP)-3	(80H-CBP)-2
CBP-2	2	(80H-CBP)-2	(80H-CBP)-1
		(80H-CBP)-1	(80H-CBP)

The eighth bit of the Input and Output FIFO Threshold SFR indicates the status of the service requests regardless of the freeze condition. If the eighth bit is a "1", the FIFO is requesting service from the external Host. In other words, when the Threshold SFR value goes below zero (2's complement), a service request is generated*. Normally the ITHR SFR is decremented after each external Host write to the Input FIFO and incremented after each internal CPU read of the Input FIFO. The OTHR SFR is decremented by internal CPU writes and incremented by external Host reads. Thus if the pointers are moved can be used to calculate the offset for the Threshold SFRs. It is best to change the Threshold SFRs only when the FIFO's are empty to avoid this complication. The threshold registers should also be updated after the pointers have been manipulated.

NOTE:

When programming the ITHR SFR, the eighth bit should be set to 1 (OR'd with 80H). This causes HSTAT SFR HST0 = 0, Input FIFO Request For Service. If ITHR bit 7 = 0 then HSTAT HST0 = 1, Input FIFO Does Not Request Service, and no interrupt will be generated.

*The 8th bit of the ITHR SFR must be set during initialization if the Host interrupt request is desired immediately upon leaving Freeze Mode.

Host Status SFR (HSTAT)

When in FIFO DMA Freeze Mode, some bits in the Host Status SFR are forced high and will not reflect the new status until the system returns to normal operation. The definition of the register in FIFO DMA Freeze Mode is as follows:

NOTE:

The internal CPU reads this shadow latch value when reading the Host Status SFR. The shadow latch will keep the information for these bits so normal operation can be resumed with the right status. The following bits are set (= 1) when FIFO DMA Freeze Mode is invoked;

HST7 Output FIFO Error Condition Flag

1 = No error.

0 = An invalid read has been done on the output FIFO or the Immediate Command Out Register by the host CPU.

NOTE:

The normal underrun error condition status is disabled. If an Immediate Command Out (IMOUT) SFR read is attempted during FIFO DMA Freeze Mode, the contents of the IMOUT SFR is output on the Data Buffer and the error status is cleared (= 0).

HST6 Immediate Command Out SFR Status

During normal operation, this bit is cleared (=0) when the IMOUT SFR is written by the UPI-452 internal CPU and set (= 1) when the IMOUT SFR is read by the external Host. Once the host-slave interface is frozen (i.e. SST5 = 0), this bit will be read as a 1 by the host CPU. A shadow latch will keep the information for this bit so normal operation can be resumed with the correct status.

Shadow latch:

1 = Internal CPU reads the IMOUT SFR

0 = Internal CPU writes to the IMOUT SFR

HST5 Data Stream Command at Output FIFO

This bit is forced to a 1 during FIFO DMA Freeze Mode to prevent the external host CPU from trying to read the DSC. Once normal operation is resumed, HST5 will reflect the Data/Command status of the current byte in the Output FIFO.

Shadow Latch (read by the internal CPU):

1 = No Data Stream Command (DSC)

0 = Data Stream Command at Output FIFO

HST4 Output FIFO Service Request Status

When FIFO DMA Freeze Mode is invoked, this bit no longer reflects the Output FIFO Request Service Status. This bit wll be forced to a "1".

HST3 Input FIFO Error Condition Flag

1 = No error.

0 = One of the following operations has been attempted by the external host and is invalid:

1) Write into the Input FIFO
2) Write into the Host Control SFR
3) Write into the Immediate Command In SFR

NOTE:
The normal Input FIFO overrun condition is disabled.

HST2 Immediate Command In SFR Status

This bit is normally cleared when the internal CPU reads the IMIN SFR and set when the external host CPU writes into the IMIN SFR. When the host-slave interface is frozen, reading and writing of the IMIN by the internal CPU will change the shadow latch of this bit. This bit will be read as a "1" by the external Host.

Shadow latch.

1 = Internal CPU writes into IMIN SFR

0 = Internal CPU reads the IMIN SFR

HST1 FIFO DMA Freeze Mode Status

1 = FIFO DMA Freeze Mode.

0 = Normal Operation (non-FIFO DMA Freeze Mode).

NOTE:
This bit is used to indicate to the external Host that the host-slave interface has been frozen and hence the external Host functions are now reduced as shown in Table 8.

HST0 Input FIFO Request Service Satus

When slave interface is frozen this bit no longer reflects the Input FIFO Request Service Status. This bit will be forced to a "1".

Slave Status SFR (SSTAT)

The Slave Status SFR is a read-only SFR. However, once the slave interface is frozen, most of the bits of this SFR can be changed by the internal CPU by reconfiguring the FIFO and accessing the FIFO Special Function Registers.

SST7 Output FIFO Overrun Error Flag

Inoperative in FIFO DMA Freeze Mode.

SST6 Immediate Command Out SFR Status

In FIFO DMA Freeze Mode, this bit will be cleared when the internal CPU reads the Immediate Command Out SFR and set when the internal CPU writes to the Immediate Command Out Register.

SST5 FIFO-External Interface FIFO DMA Freeze Mode Status

This bit indicates to the internal CPU that FIFO DMA Freeze Mode is in progress and that it has write access to the FIFO Control, Host control and Immediate Command SFRs.

SST4 Output FIFO Request Service Status

During normal operation, this bit indicates to the internal CPU that the Output FIFO is ready for more data. The status of this bit reflects the position of the Output FIFO read and write pointers. Hence, in FIFO DMA Freeze Mode, this flag can be changed by the internal CPU indirectly as the read and write pointers change.

SST3 Input FIFO Underrun Flag

Inoperative during FIFO DMA Freeze Mode.

During normal operation, a read operation clears (=0) this bit when there are no data bytes in the Input FIFO and deactivated (=1) when the Slave Status SFR is read. In FIFO DMA Freeze Mode, this bit will not be cleared by an Input FIFO read underrun error condition, nor will it be reset by the reading of the Slave Status SFR.

SST2 Immediate Command In SFR Status

This bit is normally activated (=0) when the external host CPU writes into the Immediate Command In SFR and deactivated (=1) when it is read by the internal CPU. In FIFO DMA Freeze Mode, this bit will not be activated (=0) by the external Host's writing of the Immediate Command IN SFR since this function is disabled. However, this bit will be cleared (=0) if the internal CPU writes to the Immediate Command In SFR and it will be set =1) if it reads from the register.

SST1 Data Stream Command at Input FIFO Flag

In FIFO DMA Freeze Mode, this bit operates normally. It indicates whether the next byte of data from the Input FIFO is a DSC or data byte. If it is a DSC byte, reading from the FIFO IN SFR will result in reading invalid data (FFH) and vice versa. In FIFO DMA Freeze Mode, this bit still reflects the type of data byte available from the Input FIFO.

SST0 Input FIFO Service Request Flag

During normal operation, this bit is activated (=0) when the Input FIFO contains bytes that can be read by the internal CPU and deactivated (=1) when the Input FIFO does not need any service from the internal CPU. In FIFO DMA Freeze Mode, the status of this bit should not change unless the pointers of the Input FIFO are changed. In this mode, the internal CPU can indirectly change this bit by changing the read and write pointers of the Input FIFO but cannot change it directly.

Immediate Command In/Out SFR (IMIN/IMOUT)

If FIFO DMA Freeze Mode is in progress, writing to the Immediate Command In SFR by the external host will be disabled, and any such attempt will cause HST3 to be cleared (=0). Similarly, the Immediate Command Out SFR read operation (by the host) will be disabled internally and read attempts will cause HST7 to be cleared (=0).

Internal CPU Read and Write of the FIFO During FIFO DMA Freeze Mode

In normal operation, the Input FIFO can only be read by the internal CPU and similarly, the Output FIFO can only be written by the internal CPU. During FIFO DMA Freeze Mode, the internal CPU can read the entire contents of the Input FIFO by programming the CBP SFR to 7FH, setting the IRPR SFR to zero, and then the IWPR SFR to zero. Programming the pointer registers in this order generates a FIFO full signal to the FIFO logic and enables internal CPU read operations. If the IWPR and IRPR are already zero, the write pointer should be changed to a non-zero value to clear the empty status then the pointers can be set to zero. Writing to the IRDR SFR automatically updates the look ahead registers.

In a similar manner, the internal CPU can write to all 128 bytes of the FIFO by setting the CBP SFR to zero, setting OWPR SFR to zero, and then setting ORPR SFR to zero. This generates a FIFO empty signal and allows internal CPU write operations to all 128 bytes of the FIFO. The Threshold registers also need to be adjusted when the pointers are changed. (See "Input and Output FIFO Threshold SFR" section below.)

MEMORY ORGANIZATION

The UPI-452 has separate address spaces for Program Memory and Data Memory like the 80C51. The Program Memory can be up to 64K bytes. The lower 8K of Program Memory may reside on-chip. The Data Memory consists of 256 bytes of on-chip RAM, up to 64K bytes of off-chip RAM and a number of "SFRs" (Special Function Registers) which appear as yet another set of unique memory addresses.

Table 11a. Internal Memory Addressing

Memory Space	Addressing Method
Lower 128 Bytes of Internal RAM	Direct or Indirect
Upper 128 Bytes of Internal RAM	Indirect Only
UPI-452 SFR's	Direct Only

The 80C51 Special Function Registers are listed in Table 11a, and the additional UPI-452 SFRs are listed in Table 11b. A brief description of the 80C51 core SFRs is also provided below.

Accessing External Memory

As in the 80C51, accesses to external memory are of two types: Accesses to external Program Memory and accesses to external Data Memory.

External Program Memory is accessed under two conditions:

1) Whenever signal $\overline{EA}$ = 0; or
2) Whenever the program counter (PC) contains a number that is larger than 1FFFH.

This requires that the ROMless versions have $\overline{EA}$ wired low to enable the lower 8K program bytes to be fetched from external memory.

External Data Memory is accessed using either the MOVX @DPTR (16 bit address) or the MOVX @Ri (8 bit address) instructions, or during external data memory transfers.

Table 11b. 80C51 Special Function Registers

Symbol	Name	Address	Contents
*ACC	Accumulator	0E0H	00H
*B	B Register	0F0H	00H
*PSW	Program Status Word	0D0H	00H
SP	Stack Pointer	81H	07H
DPTR	Data Pointer (consisting of DPH and DPL)	82H	0000H
*P0	Port 0	80H	0FFH
*P1	Port 1	90H	0FFH
*P2	Port 2	0A0H	0FFH
*P3	Port 3	0B0H	0FFH
*IP	Interrupt Priority Control	0B8H	0E0H
*IE	Interrupt Enable Control	0A8H	60H
TMOD	Timer/Counter Mode Control	89H	00H
*TCON	Timer/Counter Control	88H	00H
TH0	Timer/Counter 0 (high byte)	8CH	00H
TL0	Timer/Counter 0 (low byte)	8AH	00H
TH1	Timer/Counter 1 (high byte)	8DH	00H
TL1	Timer/Counter 1 (low byte)	8BH	00H
*SCON	Serial Control	98H	00H
SBUF	Serial Data Buff	99H	I
PCON	Power Control	87H	I0H

I = Indeterminate

Table 11c. UPI-452 Additional Special Function Registers

Symbol	Name	Address	Contents
BCRL0	DMA Byte Count Low Byte/	0E2H	I
BCRH0	High Byte/ Channel 0	0E3H	I
BCRL1	Low Byte/	0F2H	I
BCRH1	Hi Byte/ Channel 1	0F3H	I
CBP	Channel Boundary Pointer	0ECH	40H
CIN	COMMAND IN	0EFH	I
COUT	COMMAND OUT DMA Destination Address	0FFH	I

Table 11c. UPI-452 Additional Special Function Registers (Continued)

Symbol	Name	Address	Contents
DARL0	Low Byte/	0C2H	I
DARH0	Hi Byte/ Channel 0	0C3H	I
DARL1	Low Byte/	0D2H	I
DARH1	Hi Byte/ Channel 1	0D3H	I
DCON0	DMA0 Control	92H	00H
DCON1	DMA1 Control	93H	00H
FIN	FIFO IN	0EEH	I
FOUT	FIFO OUT	0FEH	I
HCON	Host Control	0E7H	00H
HSTAT	Host Status	0E6H	0FBH
*IEP	Interrupt Enable and Priority	0F8H	0C0H
IMIN	Immediate Command In	0FCH	I
IMOUT	Immediate Command Out	0FDH	I
IRPR	Input Read Pointer	0EBH	00H
ITHR	Input FIFO Threshold	0F6H	80H
IWPR	Input Write Pointer	0EAH	00H
MODE	Mode Register	0F9H	8FH
ORPR	Output Read Pointer	0FAH	40H
OTHR	Output FIFO Threshold	0F7H	01H
OWPR	Output Write Threshold	0FBH	40H
*P4	Port 4 DMA Source Address	0C0H	0FFH
SARL0	Low Byte/	0A2H	I
SARH0	Hi Byte/ Channel 0	0A3H	I
SARL1	Low Byte/	0B2H	I
SARH1	Hi Byte/ Channel 1	0B3H	I
*SLCON	Slave Control	0E8H	04H
SSTAT	Slave Status	0E9H	08FH

I = Indeterminate

The SFRs marked with an asterisk (*) are both bit- and byte- addressable. The functions of the SFRs are as follows:

Miscellaneous Special Function Register Description

80C51 SFRs

ACCUMULATOR

ACC is the Accumuator SFR. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

B REGISTER

The B SFR is used during multiply and divide operations. For other instructions it can be treated as another scratch pad regster.

PROGRAM STATUS WORD

The PSW SFR contains program status information as detailed in Table 12.

STACK POINTER

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

DATA POINTER

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

PORTS 0 TO 4

P0, P1, P2, P3 and P4 are the SFR latches of Ports 0, 1, 2, 3 and 4, respectively.

SERIAL DATA BUFFER

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

TIMER/COUNTER SFR

Register pairs (TH0, TL0), and (TH1, TL1) are the 16-bit counting registers for Timer/Counters 0 and 2.

POWER CONTROL SFR (PCON)

The PCON Register (Table 13) controls the power down and idle modes in the UPI-452, as well as providing the ability to double the Serial Channel baud rate. There are also two general purpose flag bits available to the user. Bits 5 and 6 are used to set the HOLD/HOLD Acknowledge mode (see "General Purpose DMA Channels" section), and bit 4 is not used.

Table 12. Program Status Word

Symbolic Address									Physical Address
PSW	CY	AC	F0	RS1	RS0	OV	—	P	0D0H
	(MSB)							(LSB)	

Symbol	Position	Name
CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry (For BCD operations)
F0	PSW.5	Flag 0 (user assignable)
RS1	PSW.4	Register Bank Select bit 1*
RS0	PSW.3	Register Bank Select bit 0*
OV	PSW.2	Overflow Flag
—	PSW.1	(reserved)
P	PSW.0	Parity Flag

*(RS1, RS0) enable internal RAM register banks as follows:

RS1	RS0	Internal RAM Register Bank
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

Table 13. PCON Special Function Register

Symbolic Address									Physical Address
PCON	SMOD	ARB	REQ	—	GF1	GF0	PD	IDL	087H
	(MSB)							(LSB)	

Symbol	Position	Function
SMOD	PCON7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either Mode 1, 2 or 3.
ARB	PCON6	HLD/HLDA Arbiter control bit *
REQ	PCON5	HLD/HLDA Requestor control bit *
—	PCON4	(reserved)
GF1	PCON3	General-purpose flag bit
GF0	PCON2	General-purpose flag bit
PD	PCON1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON0	Idle Mode bit. Setting this bit activates idle mode operation.

*See "Ext. Memory DMA" description.

NOTE:
If 1's are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (000X0000).

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C†
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin to V_{SS} .....−0.5V to V_{CC} + 0.5V
Voltage on V_{CC} to V_{SS} .....−0.5V to +6.5V
Power Dissipation.....1.0W**
V_{CC}/V_{PP} Supply Voltage with Respect to Ground During Programming .....−0.6V to +14.0V

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5V ±10%; V_{SS} = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage (except XTAL1, RST)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage (XTAL1, RST)	3.9	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage (Ports 1, 2, 3, 4)		0.45	V	I_{OL} = 1.6 mA (Note 1)
V_{OL1}	Output Low Voltage (except Ports 1, 2, 3, 4)		0.45	V	I_{OL} = 3.2 mA (Note 1)
V_{OH}	Output High Voltage (Ports 1, 2, 3, 4)	2.4		V	I_{OH} = −60 μA, V_{CC} = 5V ±10%
		0.75 V_{CC}		V	I_{OH} = −25 μA
		0.9 V_{CC}		V	I_{OH} = −10 μA
V_{OH1}	Output High Voltage (except Ports 1, 2, 3, 4 and Host Interface (Slave) Port)	2.4		V	I_{OH} = −400 μA, V_{CC} = 5V ±10%
		0.75 V_{CC}		V	I_{OH} = −150 μA
		0.9 V_{CC}		V	I_{OH} = −40 μA (Note 2)
V_{OH2}	Output High Voltage (Host Interface (Slave) Port)	2.4		V	I_{OH} = −400 μA, V_{CC} = 5V ±10%
		3.0		V	I_{OH} = 1 mA
		V_{CC} − 0.4		V	I_{OH} = −10 μA
I_{IL}	Logical 0 Input Current (Ports 1, 2, 3, 4)		−50	μA	V_{IN} = 0.45V
I_{TL}	Logical 1 to 0 Transition Current (Ports 1, 2, 3, 4)		−650	μA	V_{IN} = 2V

† Ambient Temperature under Bias for 87C452P is 0°C to 50°C.

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5V ±10%; V_{SS} = 0V (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{LI}	Input Leakage Current (except Ports 1, 2, 3, 4)		±10	μA	$0.45V < V_{IN} < V_{CC}$
I_{OZ}	Output Leakage Current (except Ports 1, 2, 3, 4)		±10	μA	$0.45V < V_{OUT} < V_{CC}$
I_{CC_1}	Operating Current (Note 6)		15	mA	V_{CC} = 5.5V, 16 MHz
I_{CC}	Operating Current (Note 7)		50	mA	V_{CC} = 5.5V, 16 MHz (Note 4)
I_{CCI}	Idle Mode Current		25	mA	V_{CC} = 5.5V, 16 MHz (Note 5)
I_{PD}	Power Down Current		100	μA	V_{CC} = 2V (Note 3)
RRST	Reset Pulldown Resistor	50	150	KΩ	
CIO	Pin Capacitance		20	pF	1 MHz, T_A = 25°C (sampled, not tested on all parts)

NOTES:
1. Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the V_{OLS} of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.
2. Capacitive loading on Ports 0 and 2 may cause the V_{OH} on ALE and PSEN to momentarily fall before the 0.9 V_{CC} specification when the address bits are stabilizing.
3. Power DOWN I_{CC} is measured with all output pins disconnected; EA = Port 0 = V_{CC}; XTAL2 N.C.; RST = V_{SS}; DB = V_{CC}; $\overline{WR}$ = $\overline{RD}$ = $\overline{DACK}$ = $\overline{CS}$ = A0 = A1 = A2 = V_{CC}. Power Down Mode is not supported on the 87C452P.
4. I_{CC} is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 5 ns, V_{IL} = V_{SS} + 0.5V, V_{IH} = V_{CC} − 0.5V; XTAL2 N.C.; EA = RST = Port 0 = V_{CC}; $\overline{WR}$ = $\overline{RD}$ = $\overline{DACK}$ = $\overline{CS}$ = A0 = A1 = A2 = V_{CC}. I_{CC} would be slightly higher if a crystal oscillator is used.
5. Idle I_{CC} is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 5 ns, V_{IL} = V_{SS} + 0.5V, V_{IH} = V_{CC} − 0.5V; XTAL2 N.C.; Port 0 = V_{CC}; EA = RST = V_{SS}; $\overline{WR}$ = $\overline{RD}$ = $\overline{DACK}$ = $\overline{CS}$ = A0 = A1 = A2 = V_{CC}.
6. 87C452P Piggyback EPROM only.
7. 80C452 and 83C452 only.

EXPLANATION OF THE AC SYMBOLS

Each timing symbol has 5 characters. The first character is always a 'T' (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list of all the characters and what they stand for:

A: Address.
C: Clock.
D: Input data.
H: Logic level HIGH.
I: Instruction (program memory contents).
L: Logic level LOW, or ALE.
P: PSEN.
Q: Output data.
R: READ signal.
T: Time.
V: Valid.
W: WRITE signal.
X: No longer a valid logic level.
Z: Float.

EXAMPLE

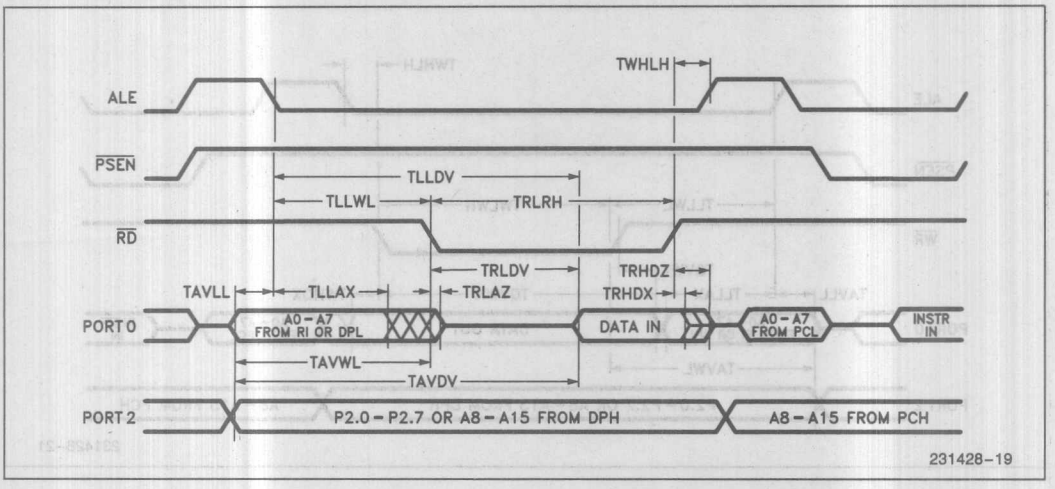
TAVLL = Time for Address Valid to ALE Low.
TLLPL = Time for ALE Low to $\overline{PSEN}$ Low.

Port 0, ALE, and PSEN = 100 pF, Load Capacitance for All Other Outputs = 80 pF

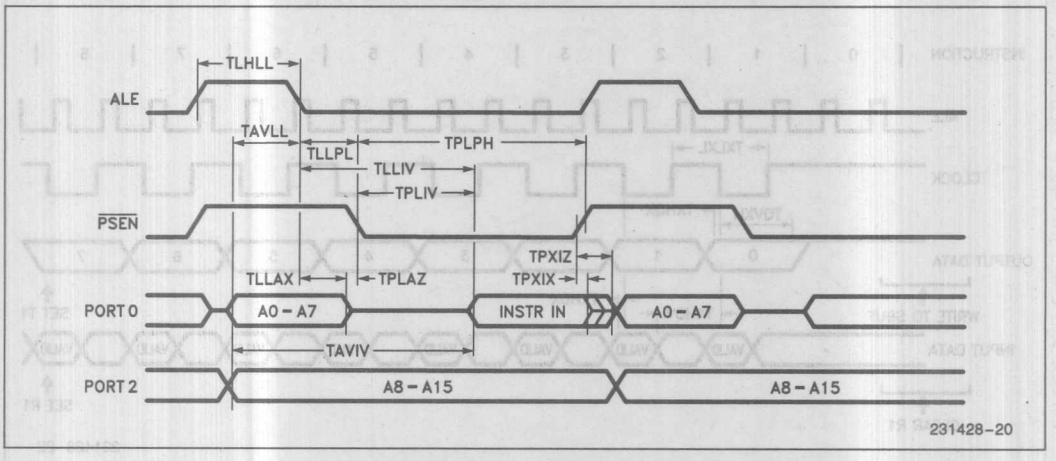
EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS

Symbol	Parameter	16 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency	3.5	16			MHz
TLHLL	ALE Pulse Width	85		2TCLCL−40		ns
TAVLL	Address Valid to ALE Low	25		TCLCL−55		ns
TLLAX	Address Hold after ALE Low	28		TCLCL−35		ns
TLLIV	ALE Low to Valid Instr In		185		4TCLCL−100	ns
TLLPL	ALE Low to PSEN Low	22		TCLCL−40		ns
TPLPH	PSEN Pulse Width	142		3TCLCL−45		ns
TPLIV	PSEN Low to Valid Instr In		110		3TCLCL−105	ns
TPXIX	Input Instr Hold after PSEN	0		0		ns
TPXIZ	Input Instr Float after PSEN		57		TCLCL−25	ns
TAVIV	Address to Valid Instr In		225		5TCLCL−105	ns
TPLAZ	PSEN Low to Address Float		10		10	ns
TRLRH	RD Pulse Width	275		6TCLCL−100		ns
TWLWH	WR Pulse Width	275		6TCLCL−100		ns
TRLDV	RD Low to Valid Data In		148		5TCLCL−165	ns
TRHDX	Data Hold after RD	0		0		ns
TRHDZ	Data Float after RD				2TCLCL−70	ns
TLLDV	ALE Low to Valid Data In		350		8TCLCL−150	ns
TAVDV	Address to Valid Data In		398		9TCLCL−165	ns
TLLWL	ALE Low to RD or WR Low	137	237	3TCLCL−50	3TCLCL+50	ns
TAVWL	Address Valid to Read or Write Low	120		4TCLCL−130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	2		TCLCL−60		ns
TWHQX	Data Hold after $\overline{WR}$	12		TCLCL−50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	23	103	TCLCL−40	TCLCL+40	ns
TQVWH	Data Valid to $\overline{WR}$ (Setup Time)	288		7TCLCL−150		ns

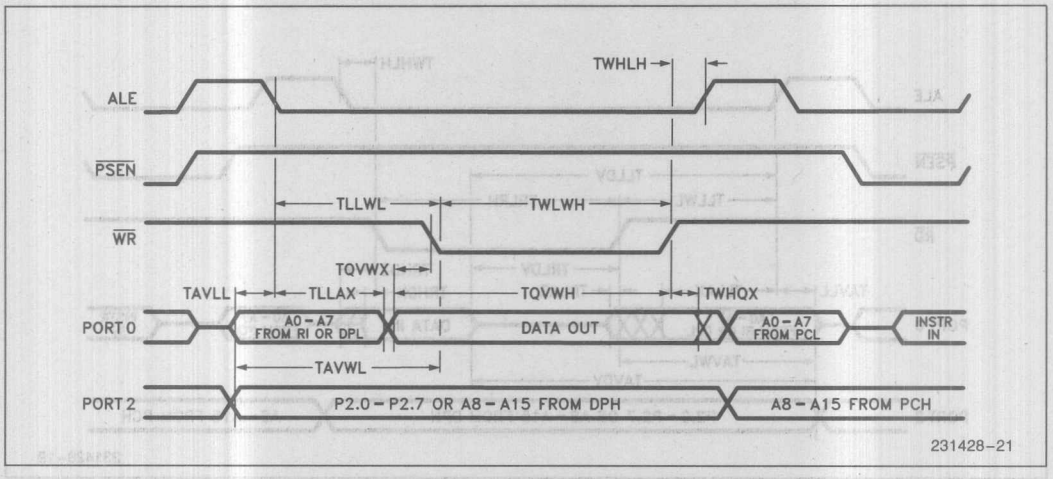
EXTERNAL DATA MEMORY READ CYCLE



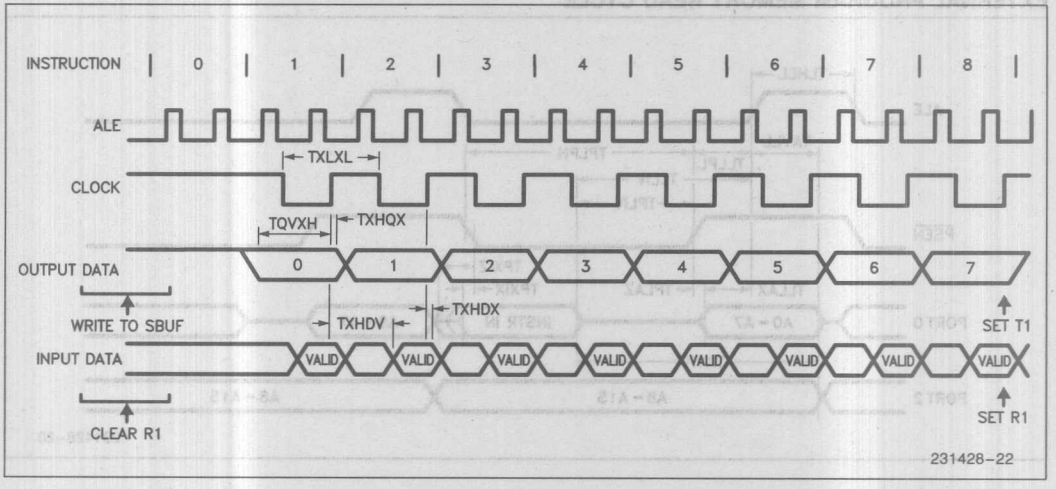
EXTERNAL PROGRAM MEMORY READ CYCLE



EXTERNAL DATA MEMORY WRITE CYCLE



SHIFT REGISTER MODE TIMING WAVEFORMS



EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Frequency	3.5	16	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

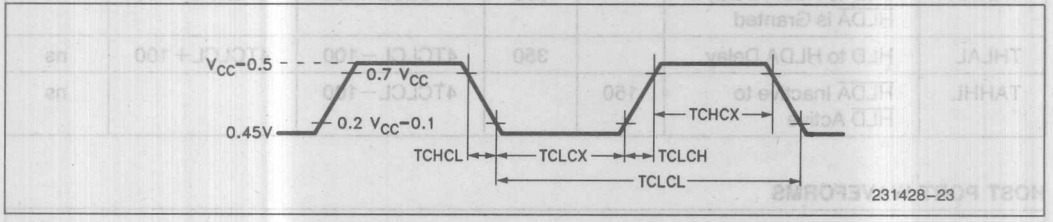
NOTE:
External clock timings are sampled, not tested on all parts.

SERIAL PORT TIMING—SHIFT REGISTER MODE

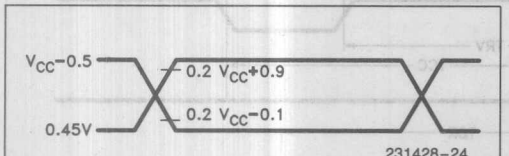
Test Conditions: T_A = 0°C to 70°C; V_{CC} = 5V ±10%; V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	16 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TXLXL	Serial Port Clock Cycle Time	750		12TCLCL		ns
TQVXH	Output Data Setup to Clock Rising Edge	492		10TCLCL−133		ns
TXHQX	Output Data Hold after Clock Rising Edge	8		2TCLCL−117		ns
TXHDX	Input Data Hold after Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		492		10TCLCL−133	ns

EXTERNAL CLOCK DRIVE WAVEFORM

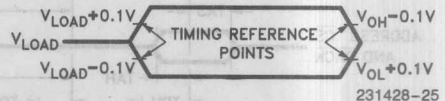


AC TESTING INPUT, OUTPUT WAVEFORMS



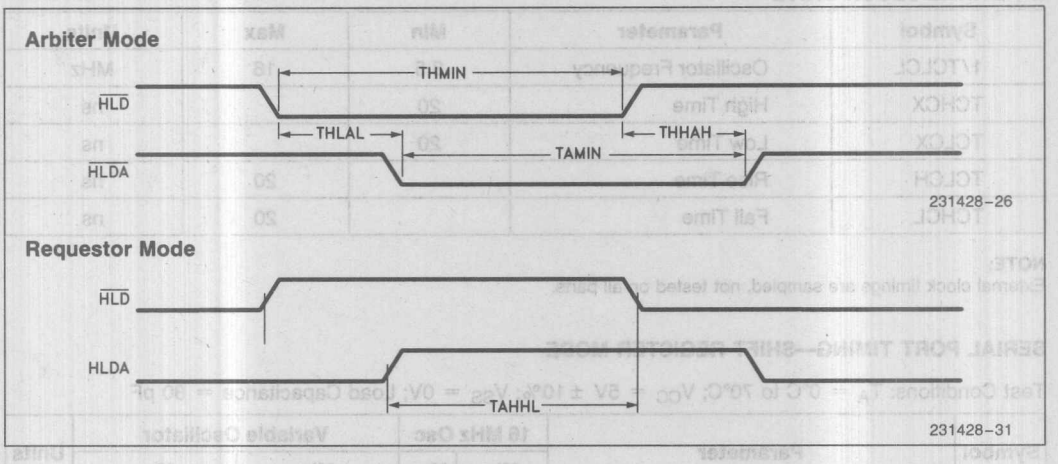
AC inputs during testing are driven at V_{CC} −0.5V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at V_{IH} min. for a logic "1" and V_{IL} max. for a logic "0".

FLOAT WAVEFORMS



For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs, and begins to float when a 100 mV change from the loaded V_{OH}/V_{OL} level occurs. $I_{OL}/I_{OH} \geq \pm 20$ mA.

HLD/HLDA WAVEFORMS

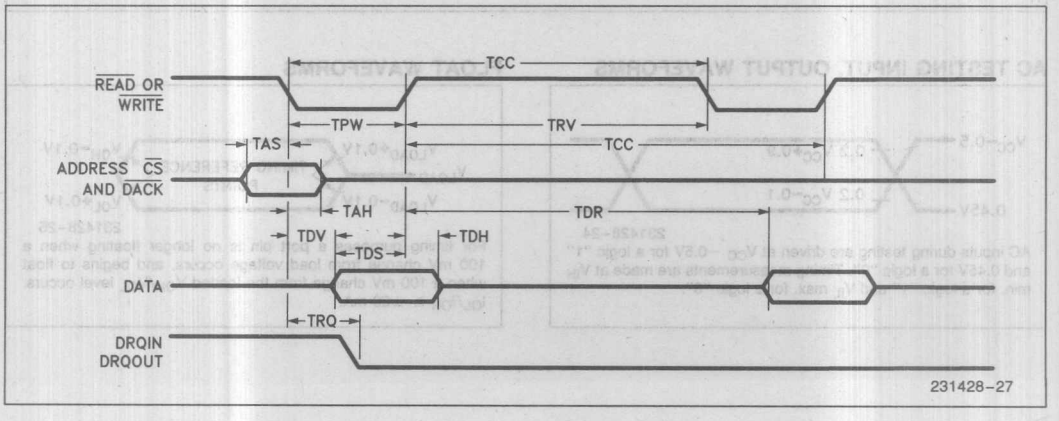


HLD/HLDA TIMINGS

Test Conditions: T_A = 0°C to +70°C; V_{CC} = 5V ±10%, V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	16 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
THMIN	$\overline{HLD}$ Pulse Width	350		4TCLCL+100		ns
TAMIN	$\overline{HLDA}$ Pulse Width	350		4TCLCL+100		ns
THHAH	$\overline{HLD}$ to $\overline{HLDA}$ Delay if $\overline{HLDA}$ is Granted		350	4TCLCL−100	4TCLCL+100	ns
THLAL	HLD to HLDA Delay		350	4TCLCL−100	4TCLCL+100	ns
TAHHL	$\overline{HLDA}$ Inactive to $\overline{HLD}$ Active	150		4TCLCL−100		ns

HOST PORT WAVEFORMS



HOST PORT TIMINGS

Test Conditions: T_A = 0°C to 70°C; V_{CC} = 5V ±10%; V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	16 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TCC	Cycle Time	375		6TCLCL		ns
TPW	Command Pulse Width	100		100		ns
TRV	Recovery Time	60		60		ns
TAS	Address Setup Time	5		5		ns
TAH	Address Hold Time	30		30		ns
TDS	Write Data Setup Time	30		30		ns
TDHW	Write Data Hold Time	5		5		ns
TDHR	Read Data Hold Time	5	40	5	40	ns
TDV	$\overline{\text{READ}}$ Active to Read Data Valid Delay	85		92 ns on 12 MHz Part		ns
TDR	$\overline{\text{WRITE}}$ Inactive to Read Data Valid Delay (Applies only to Host Control SFR)		300		4.8TCLCL	ns
TRQ	$\overline{\text{READ}}$ or $\overline{\text{WRITE}}$ Active to DRQIN or DRQOUT Delay		150		150	ns

PROGRAMMING MODES

Caution: Exceeding 14V on V_{PP} will permanently damage the device.

Initially, all bits of the EPROM are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light exposure (Cerdip EPROMs).

This device is in the programming mode when V_{PP} is raised to its programming voltage and ALE/$\overline{\text{PGM}}$ are both at TTL-low. The data to be programmed is applied 8 bits in parallel to the Port 0 pins. The levels required for the address and data inputs are TTL. The address is applied to Port 1 and 2.

Program Verify

A verify should be performed on the programmed bits to determine that they have been correctly programmed. The verify is performed with $\overline{\text{OE}}$ at V_{IL}, $\overline{\text{PGM}}$ at V_{IH} and V_{PP} and V_{CC} at their programming voltages.

inteligent Identifier™ Mode

inteligent Identifier™ Mode is not supported on the 87C452P piggyback EPROM device.

ERASURE CHARACTERISTICS

The erasure characteristics are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000Å–4000Å range. Data shows that constant exposure to room level fluorescent lighting could erase the EPROM in approximately three years, while it would take approximately one week to cause erasure when exposed to direct sunlight. If the EPROM is to be exposed to these types of lighting conditions for extended periods of time, opaque labels should be placed over the window to prevent unintentional erasure.

The recommended erasure procedure is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of fifteen (15) Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The EPROM should be placed within one inch of the lamp tubes during erasure. The maximum integrated dose an EPROM can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 μW/cm^2). Exposure of the EPROM to high intensity UV light for longer periods may cause permanent damage.

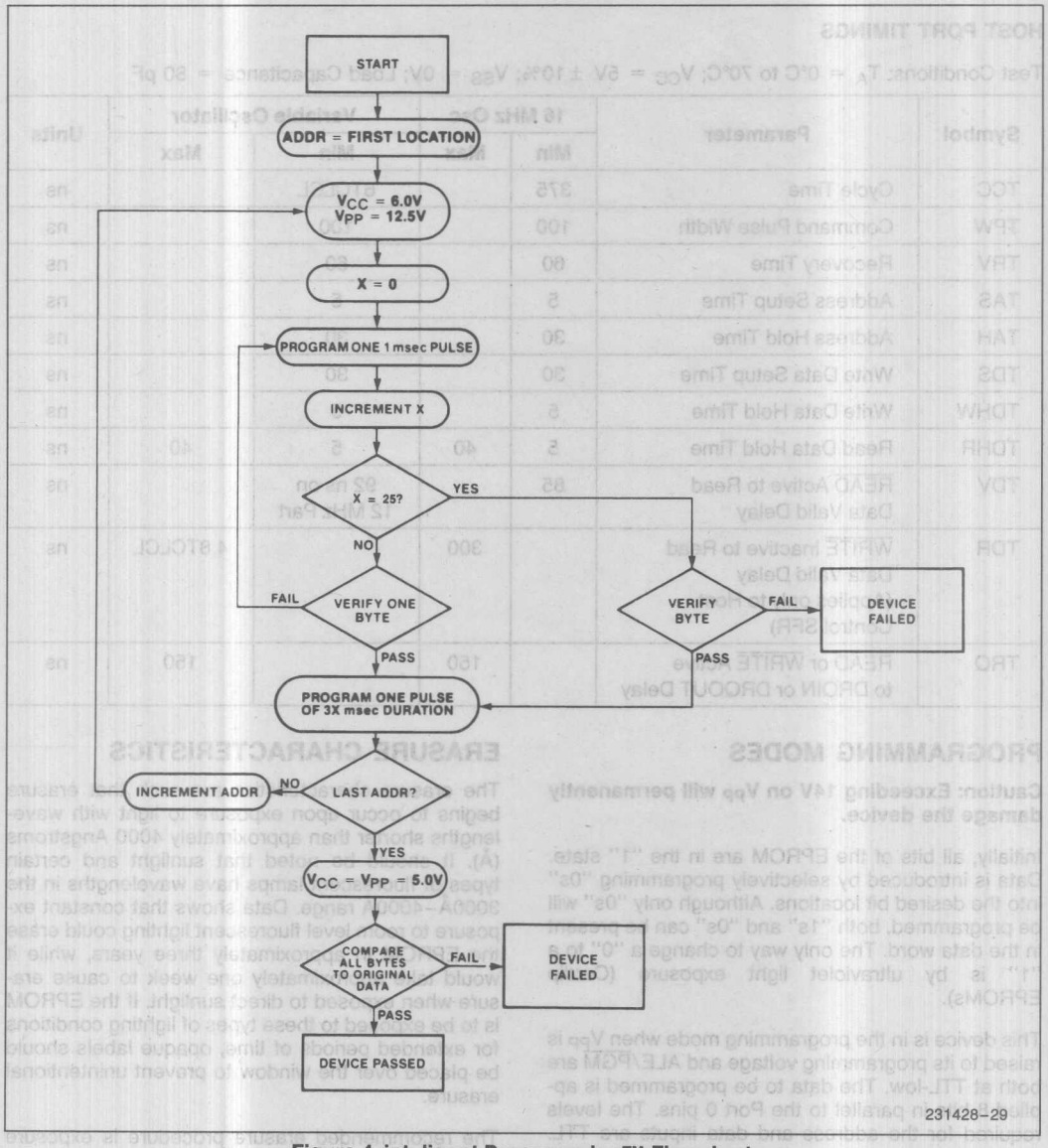


Figure 4. int$_e$ligent Programming™ Flowchart

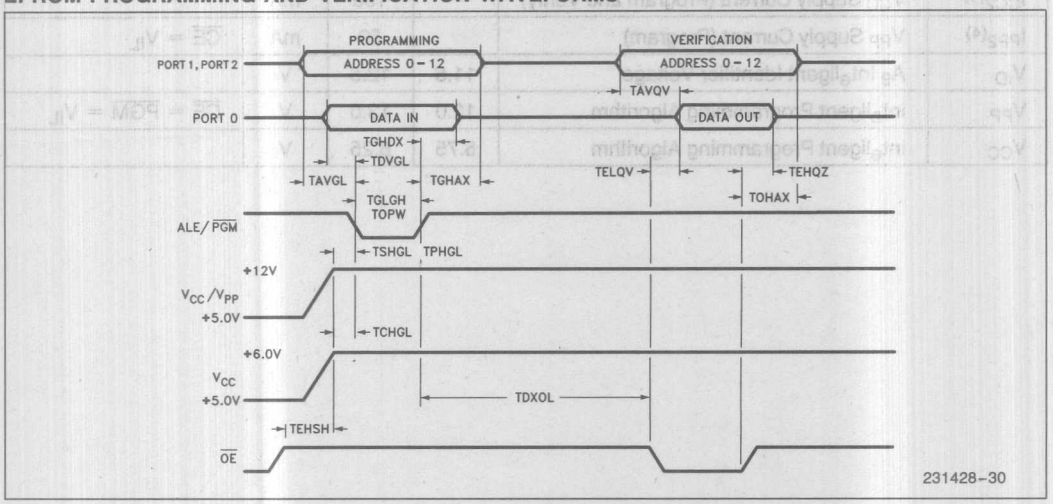
int$_e$ligent Programming™ Algorithm

The int$_e$ligent Programming Algorithm, a standard in the industry for the past few years, is required for all of Intel's 12.5V DERDEP EPROMs. Plastic amd PLCC EPROMs may also be programmed using this method. A flowchart of the int$_e$ligent Programming Algorithm is shown in Figure 4.

The int$_e$ligent Programming Algorithm utilizes two different pulse types: initial and overprogram. The duration of the initial $\overline{PGM}$ pulse(s) is one millisecond, which will then be followed by a longer overprogram pulse of length 3X ms. X is an iteration counter and is equal to the number of the initial one millisecond pulses applied to a particular location, before a correct verify occurs. Up to 25 one-millisecond pulses per byte are provided for before the overprogram pulse is applied.

The entire sequence of program pulses and byte verifications is performed at V_{CC} = 6.0V and V_{PP} = 12.5V. When the int$_e$ligent Programming cycle has been completed, all bytes should be compared to the original data with V_{CC} = V_{PP} = 5.0V.

EPROM PROGRAMMING AND VERIFICATION WAVEFORMS



A.C. PROGRAMMING CHARACTERISTICS

T_A = 25°C ±5°C (see EPROM PROGRAMMING D.C. CHARACTERISTICS for V_{CC} and V_{PP} Voltages)

Symbol	Parameter	12 MHz OSC			Test Conditions*(1)
		Min	Max	Unit	
TAVGL	Address Setup to $\overline{PGM}$		2.9	µs	
TDXOL	$\overline{OE}$ Setup from Data Float	2.0		µs	
TDVGL	Data Setup to $\overline{PGM}$	3.8		µs	
TOHAX	Address Hold after $\overline{OE}$	0		µs	
TGHDX	Data Hold after $\overline{PGM}$	2.0		ns	
TEHQZ	Data Float after $\overline{OE}$	0	1.6	µs	(Note 3)
TOHAX	Address Hold after $\overline{PGM}$	0		µs	
TPHGL	V_{PP} Setup to $\overline{PGM}$	2.0		µs	
TCHGL	V_{CC} Setup to $\overline{PGM}$	2.0		µs	
TEHSH	$\overline{OE}$ Setup to V_{PP} and V_{CC}	2.0		µs	
TGLGH	$\overline{PGM}$ Pulse Width	0.95	1.05	ms	int$_e$ligent Programming
TAVQV	Address to Data Valid		3.0	µs	
TOPW	$\overline{PGM}$ Overprogram Pulse Width	2.85	78.75	ms	(Note 2)
TELQV	Data Valid from $\overline{OE}$		2.0	µs	

NOTES:
1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
2. The length of the overprogram pulse may vary from 2.85 ms to 78.75 ms as a function of the iteration counter value X (int$_e$ligent Programming Algorithm only).
3. This parameter is only sampled and is not 100% tested. Output Float is defined as the point where data is no longer driven—see timing diagram.
4. The maximum current value is with Port 0 unloaded.

D.C. PROGRAMMING CHARACTERISTICS $T_A = 25°C \pm 5\%$

Symbol	Parameter	Limits			Test Conditions*(1)
		Min	Max	Unit	
I_{CC2}(4)	V_{CC} Supply Current (Program and Verify)		150	mA	
I_{PP2}(4)	V_{PP} Supply Current (Program)		50	mA	$\overline{CE} = V_{IL}$
V_{ID}	A_9 Int$_e$ligent Identifier Voltage	11.5	12.5	V	
V_{PP}	int$_e$ligent Programming Algorithm	12.0	13.0	V	$\overline{CE} = \overline{PGM} = V_{IL}$
V_{CC}	int$_e$ligent Programming Algorithm	5.75	6.25	V	



PRELIMINARY

UPI™-41, 42: 8041AH/8042AH/8741AH/8742AH UNIVERSAL PERIPHERAL INTERFACE 8-BIT SLAVE MICROCONTROLLER

- **UPI-41: 6 MHz; UPI-42: 12 MHz**
- **Pin, Software and Architecturally Compatible with all UPI-41 and UPI-42 Products**
- **8-Bit CPU plus ROM/EPROM, RAM, I/O, Timer/Counter and Clock in a Single Package**
- **2048 x 8 ROM/EPROM, 256 x 8 RAM on UPI-42, 1024 x 8 ROM/EPROM, 128 x 8 RAM on UPI-41, 8-Bit Timer/Counter, 18 Programmable I/O Pins**
- **One 8-Bit Status and Two Data Registers for Asynchronous Slave-to-Master Interface**
- **DMA, Interrupt, or Polled Operation Supported**
- **Fully Compatible with all Intel and Most Other Microprocessor Families**
- **Interchangeable ROM and EPROM Versions**
- **Expandable I/O**
- **Sync Mode Available**
- **Over 90 Instructions: 70% Single Byte**
- **Available in EXPRESS — Standard Temperature Range**
- **int$_e$ligent Programming™ Algorithm — Fast EPROM Programming**
- **Available in 40-Lead Cerdip, 40-Lead Plastic and 44-Lead Plastic Leaded Chip Carrier Packages**

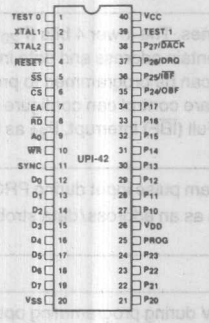
(See Packaging Spec., Order #231369)

The Intel UPI-41 and UPI-42 are general-purpose Universal Peripheral Interfaces that allow the designer to develop customized solutions for peripheral device control.

They are essentially "slave" microcontrollers, or microcontrollers with a slave interface included on the chip. Interface registers are included to enable the UPI device to function as a slave peripheral controller in the MCS™ Modules and iAPX family, as well as other 8-, 16-bit systems.

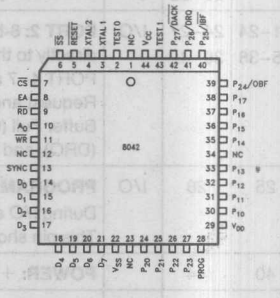
To allow full user flexibility, the program memory is available in ROM, One-Time Programmable EPROM (OTP) and UV-erasable EPROM. All UPI-41 and UPI-42 devices are fully pin compatible for easy transition from prototype to production level designs. These are the memory configurations available.

UPI Device	ROM	EPROM	RAM	Programming Voltage
8042AH	2K	—	256	—
8742AH	—	2K	256	12.5V
8041AH	1K	—	128	—
8741AH	—	1K	128	12.5V



210393-2

Figure 1. DIP Pin Configuration



210393-3

Figure 2. PLCC Pin Configuration

October 1987
Order Number: 210393-004

Table 1. Pin Description

Symbol	DIP Pin No.	PLCC Pin No.	Type	Name and Function
TEST 0, TEST 1	1 39	2 43	I	**TEST INPUTS:** Input pins which can be directly tested using conditional branch instructions. **FREQUENCY REFERENCE:** TEST 1 (T_1) also functions as the event timer input (under software control). TEST 0 (T_0) is used during PROM programming and ROM/EPROM verification. It is also used during Sync Mode to reset the instruction state to S1 and synchronize the internal clock to PH1. See the Sync Mode Section.
XTAL 1, XTAL 2	2 3	3 4	I	**INPUTS:** Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
$\overline{\text{RESET}}$	4	5	I	**RESET:** Input used to reset status flip-flops and to set the program counter to zero. $\overline{\text{RESET}}$ is also used during EPROM programming and verification.
$\overline{\text{SS}}$	5	6	I	**SINGLE STEP:** Single step input used in conjunction with the SYNC output to step the program through each instruction (EPROM). This should be tied to +5V when not used. This pin is also used to put the device in Sync Mode by applying 12.5V to it.
$\overline{\text{CS}}$	6	7	I	**CHIP SELECT:** Chip select input used to select one UPI microcomputer out of several connected to a common data bus.
EA	7	8	I	**EXTERNAL ACCESS:** External access input which allows emulation, testing and ROM/EPROM verification. This pin should be tied low if unused.
$\overline{\text{RD}}$	8	9	I	**READ:** I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.
A_0	9	10	I	**COMMAND/DATA SELECT:** Address Input used by the master processor to indicate whether byte transfer is data ($A_0 = 0$, F1 is reset) or command ($A_0 = 1$, F1 is set). $A_0 = 0$ during program and verify operations.
$\overline{\text{WR}}$	10	11	I	**WRITE:** I/O write input which enables the master CPU to write data and command words to the UPI INPUT DATA BUS BUFFER.
SYNC	11	13	O	**OUTPUT CLOCK:** Output signal which occurs once per UPI instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.
D_0–D_7 (BUS)	12–19	14–21	I/O	**DATA BUS:** Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI microcomputer to an 8-bit master system data bus.
P_{10}–P_{17}	27–34	30–33 35–38	I/O	**PORT 1:** 8-bit, PORT 1 quasi-bidirectional I/O lines. P_{10}–P_{17} access the signature row and security bit.
P_{20}–P_{27}	21–24 35–38	24–27 39–42	I/O	**PORT 2:** 8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P_{20}–P_{23}) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4–7 access. The upper 4 bits (P_{24}–P_{27}) can be programmed to provide interrupt Request and DMA Handshake capability. Software control can configure P_{24} as Output Buffer Full (OBF) interrupt, P_{25} as Input Buffer Full ($\overline{\text{IBF}}$) interrupt, P_{26} as DMA Request (DRQ), and P_{27} as DMA ACKnowledge ($\overline{\text{DACK}}$).
PROG	25	28	I/O	**PROGRAM:** Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243. This pin should be tied high if unused.
V_{CC}	40	44		**POWER:** +5V main power supply pin.
V_{DD}	26	29		**POWER:** +5V during normal operation. +12.5V during programming operation. Low power standby supply pin.
V_{SS}	20	22		**GROUND:** Circuit ground potential.

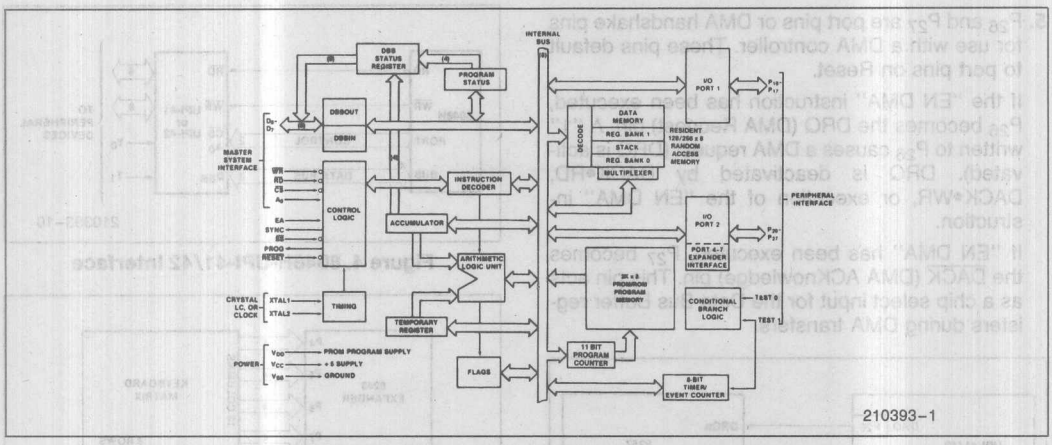
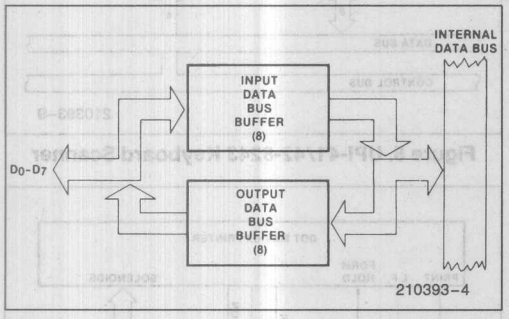


Figure 3. Block Diagram

UPI-41 and UPI-42 FEATURES

1. Two Data Bus Buffers, one for input and one for output. This allows a much cleaner Master/Slave protocol.



2. 8 Bits of Status

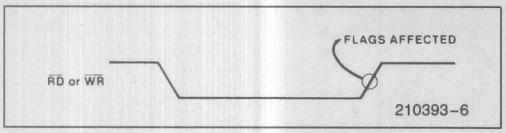
ST_7	ST_6	ST_5	ST_4	F_1	F_0	IBF	OBF
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

ST_4–ST_7 are user definable status bits. These bits are defined by the "MOV STS, A" single byte, single cycle instruction. Bits 4–7 of the acccumulator are moved to bits 4–7 of the status register. Bits 0–3 of the status register are not affected.

MOV STS, A Op Code: 90H

1	0	0	1	0	0	0	0
D_7							D_0

3. $\overline{RD}$ and $\overline{WR}$ are edge triggered. IBF, OBF, F_1 and INT change internally after the trailing edge of $\overline{RD}$ or $\overline{WR}$.

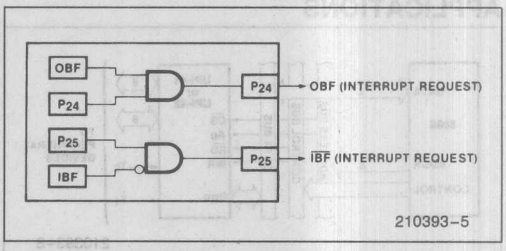


During the time that the host CPU is reading the status register, the UPI is prevented from updating this register or is 'locked out.'

4. P_{24} and P_{25} are port pins or Buffer Flag pins which can be used to interrupt a master processor. These pins default to port pins on Reset.

   If the "EN FLAGS" instruction has been executed, P_{24} becomes the OBF (Output Buffer Full) pin. A "1" written to P_{24} enables the OBF pin (the pin outputs the OBF Status Bit). A "0" written to P_{24} disables the OBF pin (the pin remains low). This pin can be used to indicate that valid data is available from the UPI (in Output Data Bus Buffer).

   If "EN FLAGS" has been executed, P_{25} becomes the $\overline{IBF}$ (Input Buffer Full) pin. A "1" written to P_{25} enables the $\overline{IBF}$ pin (the pin outputs the inverse of the IBF Status Bit. A "0" written to P_{25} disables the $\overline{IBF}$ pin (the pin remains low). This pin can be used to indicate that the UPI is ready for data.



Data Bus Buffer Interrupt Capability

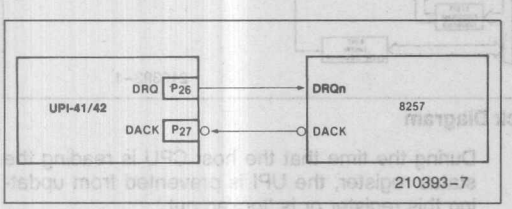
EN FLAGS Op Code: 0F5H

1	1	1	1	0	1	0	1
D_7							D_0

for use with a DMA controller. These pins default to port pins on Reset.

If the "EN DMA" instruction has been executed, P_{26} becomes the DRQ (DMA Request) pin. A "1" written to P_{26} causes a DMA request (DRQ is activated). DRQ is deactivated by DACK•RD, DACK•WR, or execution of the "EN DMA" instruction.

If "EN DMA" has been executed, P_{27} becomes the $\overline{DACK}$ (DMA ACKnowledge) pin. This pin acts as a chip select input for the Data Bus Buffer registers during DMA transfers.



DMA Handshake Capability

EN DMA Op Code: 0E5H

1	1	1	0	0	1	0	1

D_7 D_0

6. When EA is enabled on the UPI, the program counter is placed on Port 1 and the lower three bits of Port 2 (MSB = P_{22}, LSB = P_{10}). On the UPI this information is multiplexed with PORT DATA (see port timing diagrams at end of this data sheet).
7. The 8741AH and 8742AH support the intelligent Programming Algorithm. (See the Programming Section.)

APPLICATIONS

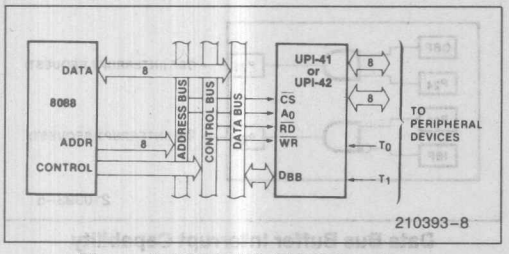


Figure 3. 8088-UPI-41/42 Interface

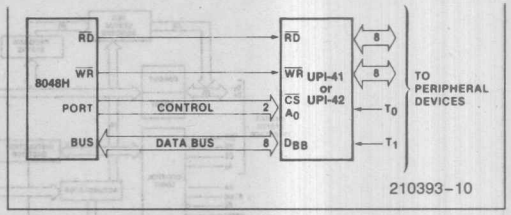


Figure 4. 8048H-UPI-41/42 Interface

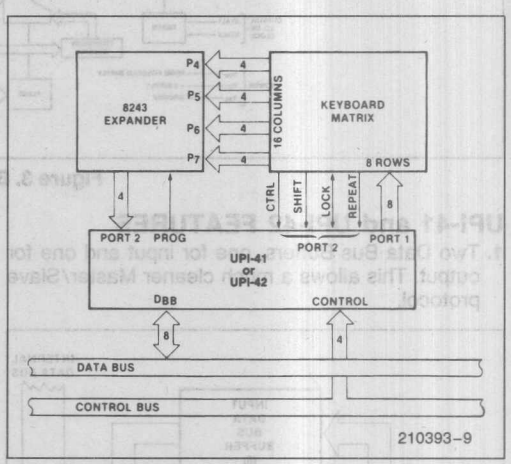


Figure 5. UPI-41/42-8243 Keyboard Scanner

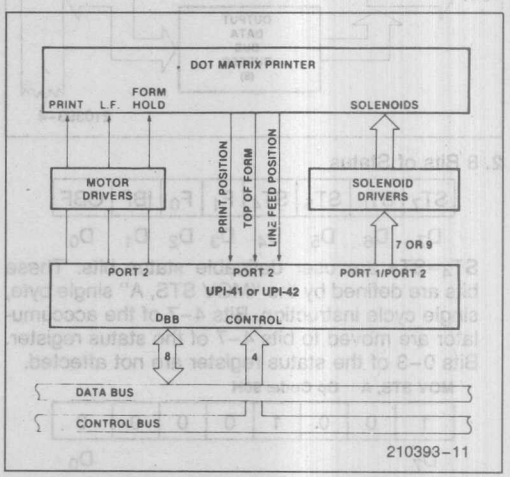


Figure 6. UPI-41/42 80-Column Matrix Printer Interface

ERASING THE 8741AH and 8742AH EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	2 Clock Inputs
$\overline{\text{Reset}}$	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Signature Row/Security Bit Modes
BUS	Address and Data Input Data Output During Verify
P_{20-22}	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING
An attempt to program a missocketed 8741AH or 8742AH will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. CS = 5V, V_{CC} = 5V, V_{DD} = 5V, RESET = 0V, A_0 = 0V, TEST 0 = 5V, clock applied or internal oscillator operating, BUS floating, PROG = 5V.
2. Insert 8741AH or 8742AH in programming socket
3. TEST 0 = 0V (select program mode)
4. EA = 12.5V (active program mode)
5. V_{CC} = 6V (programming supply)
6. V_{DD} = 12.5V (programming power)
7. Address applied to BUS and P_{20-22}
8. $\overline{\text{RESET}}$ = 5V (latch address)
9. Data applied to BUS
10. PROG = 5V followed by one 1 ms pulse to 0V
11. TEST 0 = 5V (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0V
14. Apply overprogram pulse
15. $\overline{\text{RESET}}$ = 0V and repeat from step 6
16. Programmer should be at conditions of step 1 when 8741AH or 8742AH is removed from socket

Please follow the int$_e$ligent Programming flow chart for proper programming procedure.

Characteristics

The erasure characteristics of the 8741AH/8742AH are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8741AH/8742AH in approximately 3 years while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 8741AH/8742AH is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8741AH/8742AH window to prevent unintentional erasure.

The recommended erasure procedure for the 8741AH/8742AH is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 w-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 8741AH/8742AH should be placed within one inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure. Exposure of the 8741AH/8742AH to high intensity UV light for long periods may cause permanent damage.

int$_e$ligent Programming™ Algorithm

The int$_e$ligent Programming Algorithm rapidly programs Intel 8741AH/8742AH EPROMs using an efficient and reliable method particularly suited to the production programming environment. Typical programming time for individual devices is on the order of 10 seconds. Programming reliability is also ensured as the incremental program margin of each byte is continually monitored to determine when it has been successfully programmed. A flowchart of the 8741AH/8742AH int$_e$ligent Programming Algorithm is shown in Figure 7.

The int$_e$ligent Programming Algorithm utilizes two different pulse types: initial and overprogram. The duration of the initial PROG pulse(s) is one millisecond, which will then be followed by a longer overprogram pulse of length 3X msec. X is an iteration counter and is equal to the number of the initial one millisecond pulses applied to a particular 8741AH/8742AH location, before a correct verify occurs. Up to 25 one-millisecond pulses per byte are provided for before the overprogram pulse is applied.

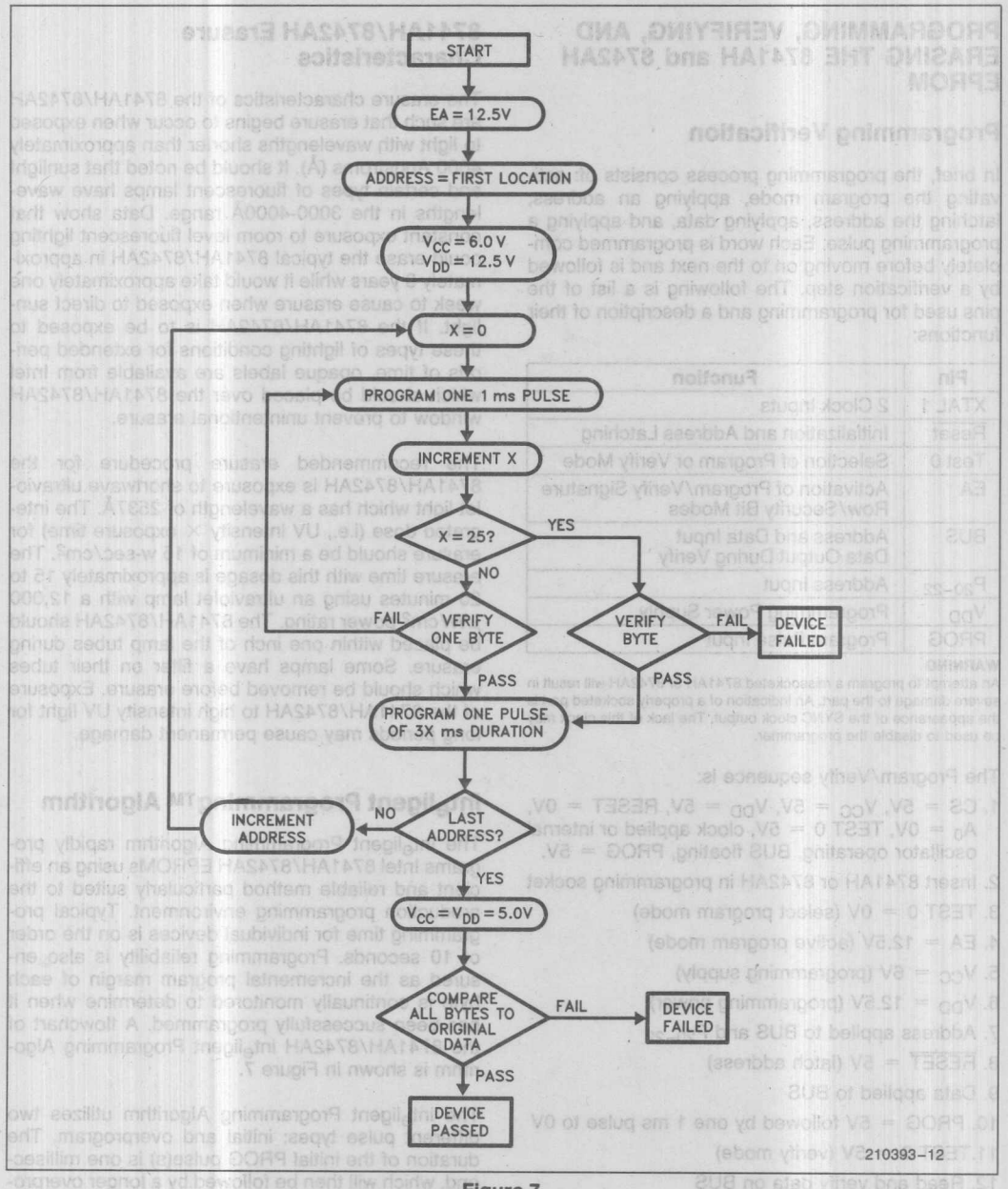

START
EA = 12.5V
ADDRESS = FIRST LOCATION
V_{CC} = 6.0 V
V_{DD} = 12.5 V
X = 0
PROGRAM ONE 1 ms PULSE
INCREMENT X
X = 25?
YES
NO
FAIL
VERIFY ONE BYTE
VERIFY BYTE
FAIL
DEVICE FAILED
PASS
PASS
PROGRAM ONE PULSE OF 3X ms DURATION
INCREMENT ADDRESS
NO
LAST ADDRESS?
YES
V_{CC} = V_{DD} = 5.0V
COMPARE ALL BYTES TO ORIGINAL DATA
FAIL
DEVICE FAILED
PASS
DEVICE PASSED
210393-12

Figure 7

The entire sequence of program pulses and byte verifications is performed at V_{CC} = 6.0V and V_{DD} = 12.5V. When the int$_e$ligent Programming cycle has been completed, all bytes should be compared to the original data with V_{CC} = 5.0, V_{DD} = 5V.

Verify

A verify should be performed on the programmed bits to determine that they have been correctly programmed. The verify is performed with T0 = 5V, V_{DD} = 5V, EA = 12.5V, $\overline{SS}$ = 5V, PROG = 5V, A0 = 0V, and $\overline{CS}$ = 5V.

SECURITY BIT

The security bit is a single EPROM cell outside the EPROM array. The user can program this bit with the appropriate access code and the normal programming procedure, to inhibit any external access to the EPROM contents. Thus the user's resident program is protected. There is no direct external access to this bit. However, the security byte in the signature mode has the same address and can be used to check indirectly whether the security bit has been programmed or not. The security bit has no effect on the signature mode, so the security byte can always be examined.

SECURITY BIT PROGRAMMING/ VERIFICATION

Programming

a. Read the security byte of the signature mode. Make sure it is 00H.

b. Apply access code to appropriate inputs to put the device into security mode.

c. Apply high voltage to EA and V_{DD} pins.

d. Follow the programming procedure as per the int$_e$ligent Programming Algorithm with known data on the databus. Not only the security bit, but also the security byte of the signature mode is programmed.

e. Verify that the security byte of the signature mode contains the same data as appeared on the data bus. (If DB0–DB7 = high, the security byte will contain FFH.)

f. Read two consecutive known bytes from the EPROM array and verify that the wrong data are retrieved in at least one verification. If the EPROM can still be read, the security bit may have not been fully programmed though the security byte in the signature mode has.

Verification

Since the security bit address overlaps the address of the security byte of the signature mode, it can be used to check indirectly whether the security bit has been programmed or not. Therefore, the security bit verification is a mere read operation of the security byte of the signature row (1 = security bit programmed; 0 = security bit unprogrammed). Note that during the security bit programming, the reading of the security byte does not necessarily indicate that the security bit has been successfully programmed. Thus, it is recommended that two consecutive known bytes in the EPROM array be read and the wrong data should be read at least once, because it is highly improbable that random data coincides with the correct ones twice.

SIGNATURE MODE

The UPI-41/42 has an additional 32 bytes of EPROM available for Intel and user signatures and miscellaneous purposes. The 32 bytes are partitioned as follows:

A. **Test code/checksum**—This can accommodate up to 25 bytes of code for testing the internal nodes that are not testable by executing from the external memory. The test code/checksum is present on ROMs, and OTPs.

B. **Intel signature**—This allows the programmer to read from the UPI-41/42 the manufacturer of the device and the exact product name. It facilitates automatic device identification and will be present in the ROM and OTP versions. Location 10H contains the manufacturer code. For Intel, it is 89H. Location 11H contains the device code. The code is 43H and 42H for the 8042AH and OTP 8742AH, and 41H and 40H for the 8041AH and OTP 8741AH, respectively.

C. **User signature**—The user signature memory is implemented in the EPROM and consists of 2 bytes for the customer to program his own signature code (for identification purposes and quick sorting of previously programmed materials).

D. **Test signature**—This memory is used to store testing information such as: test data, bin number, etc. (for use in quality and manufacturing control).

E. **Security byte**—This byte is used to check whether the security bit has been programmed (see the security bit section).

The signature mode can be accessed by setting P10 = 0, P11–P17 = 1, and then following the programming and/or verification procedures. The location of the various address partitions are as follows:

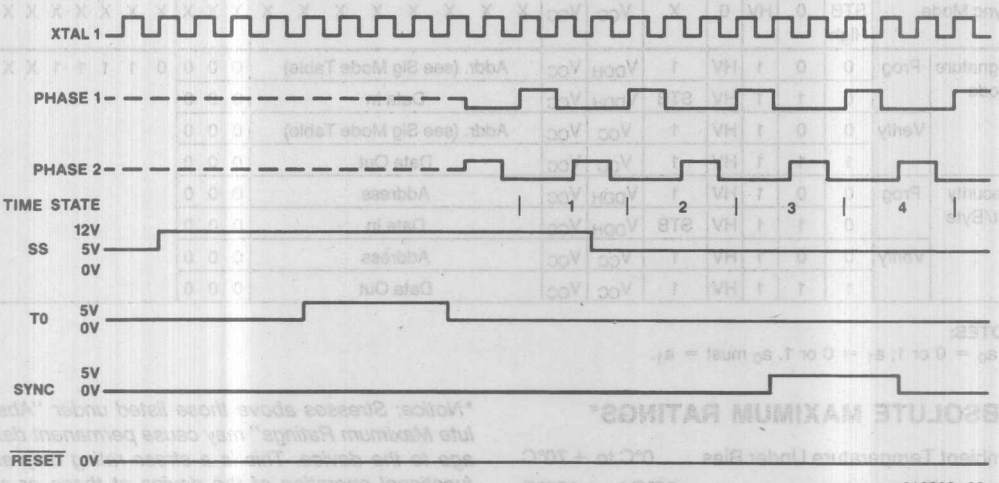
	Address		Device Type	No. of Bytes
Test Code/Checksum	0 16H	0FH 1EH	ROM/OTP	25
Intel Signature	10H	11H	ROM/OTP	2
User Signature	12H	13H	EPROM/OTP	2
Test Signature	14H	15H	ROM/OTP	2
Security Byte	1FH		EPROM/OTP	1

SYNC MODE

The UPI-41/42 has incorporated a new mode. The Sync Mode is provided to ease the design of multiple controller circuits by allowing the designer to force the device into known phase and state time. The Sync Mode may also be utilized by automatic test equipment (ATE) for quick, easy, and efficient synchronizing between the tester and the DUT (device under test).

Sync Mode is enabled when $\overline{SS}$ pin is raised to high voltage level of +12 volts. To begin synchronization, T0 is raised to 5 volts at least four clock cycles after $\overline{SS}$. T0 must be high for at least four X1 clock cycles to fully reset the prescaler and time state generators. T0 may then be brought down during low state of X1. Two clock cycles later, with the rising edge of X1, the device enters into Time State 1, Phase 1. $\overline{SS}$ is then brought down to 5 volts 4 clocks later after T0. RESET is allowed to go high 5 tCY (75 clocks) later for normal execution of code.

SYNC MODE TIMING DIAGRAMS



210393-28

Minimum Specifications
SYNC Operation Time, t_{SYNC} = 3.5 XTAL 1 Clock cycles. Reset Time, t_{RS} = 4 t_{CY}.

NOTE:
The rising and falling edges of T0 should occur during low state of XTAL1 clock.

The following table summarizes the access codes required to invoke the Sync Mode, Signature Mode, and the Security Bit, respectively. Also, the programming and verification modes are included for comparison.

Modes		Control Signals							Data Bus								Access Code									
																	Port 2			Port 1						
		T0	RST	SS	EA	PROG	V_{DD}	V_{CC}	0	1	2	3	4	5	6	7	0	1	2	0	1	2	3	4	5	6	7
Programming Mode		0	0	1	HV	1	V_{DDH}	V_{CC}	Address								Addr			a_0	a_1	X	X	X	X	X	X
		0	1	1	HV	STB	V_{DDH}	V_{CC}	Data In								Addr										
Verification Mode		0	0	1	HV	1	V_{CC}	V_{CC}	Address								Addr			a_0	a_1	X	X	X	X	X	X
		1	1	1	HV	1	V_{CC}	V_{CC}	Data Out								Addr										
Sync Mode		STB High	0	HV	0	X	V_{CC}	V_{CC}	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Signature Mode	Prog	0	0	1	HV	1	V_{DDH}	V_{CC}	Addr. (see Sig Mode Table)								0	0	0	0	1	1	1	1	X	X	1
		0	1	1	HV	STB	V_{DDH}	V_{CC}	Data In								0	0	0								
	Verify	0	0	1	HV	1	V_{CC}	V_{CC}	Addr. (see Sig Mode Table)								0	0	0								
		1	1	1	HV	1	V_{CC}	V_{CC}	Data Out								0	0	0								
Security Bit/Byte	Prog	0	0	1	HV	1	V_{DDH}	V_{CC}	Address								0	0	0								
		0	1	1	HV	STB	V_{DDH}	V_{CC}	Data In								0	0	0								
	Verify	0	0	1	HV	1	V_{CC}	V_{CC}	Address								0	0	0								
		1	1	1	HV	1	V_{CC}	V_{CC}	Data Out								0	0	0								

NOTES:
1. a_0 = 0 or 1; a_1 = 0 or 1. a_0 must = a_1.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to +70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground.....−0.5V to +7V
Power Dissipation .....1.5 W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = V_{DD} = +5V ±10%

Symbol	Parameter	8042/8742		Units	Notes
		Min	Max		
V_{IL}	Input Low Voltage (Except XTAL1, XTAL2, RESET)	−0.5	0.8	V	
V_{IL1}	Input Low Voltage (XTAL1, XTAL2, RESET)	−0.5	0.6	V	
V_{IH}	Input High Voltage (Except XTAL1, XTAL2, RESET)	2.0	V_{CC}	V	
V_{IH1}	Input High Voltage (XTAL1, RESET)	3.5	V_{CC}	V	
V_{IH2}	Input High Voltage (XTAL2)	2.2	V_{CC}	V	
V_{OL}	Output Low Voltage (D_0–D_7)		0.45	V	I_{OL} = 2.0 mA

D.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$, $V_{CC} = V_{DD} = +5V \pm 10\%$ (Continued)

Symbol	Parameter	8042/8742		Units	Notes
		Min	Max		
V_{OL1}	Output Low Voltage ($P_{10}P_{17}$, $P_{20}P_{27}$, Sync)		0.45	V	$I_{OL} = 1.6$ mA
V_{OL2}	Output Low Voltage (PROG)		0.45	V	$I_{OL} = 1.0$ mA
V_{OH}	Output High Voltage (D_0–D_7)	2.4		V	$I_{OH} = -400\ \mu A$
V_{OH1}	Output High Voltage (All Other Outputs)	2.4			$I_{OH} = -50\ \mu A$
I_{IL}	Input Leakage Current (T_0, T_1, RD, WR, CS, A_0, EA)		±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{OFL}	Output Leakage Current (D_0–D_7, High Z State)		±10	μA	$V_{SS} + 0.45 \le V_{OUT} \le V_{CC}$
I_{LI}	Low Input Load Current ($P_{10}P_{17}$, $P_{20}P_{27}$)		0.3	mA	$V_{IL} = 0.8V$
I_{LI1}	Low Input Load Current (RESET, SS)		0.2	mA	$V_{IL} = 0.8V$
I_{DD}	V_{DD} Supply Current		20	mA	Typical = 8 mA
$I_{CC} + I_{DD}$	Total Supply Current		135	mA	Typical = 80 mA
I_{DD} Standby	Power Down Supply Current		20	mA	Typical = 8 mA
I_{IH}	Input Leakage Current (P_{10}–P_{17}, P_{20}–P_{27})		100	μA	$V_{IN} = V_{CC}$
C_{IN}	Input Capacitance		10	pF	$T_A = 25°C$ (1)
C_{IO}	I/O Capacitance		20	pF	$T_A = 25°C$ (1)

NOTE:
1. Sampled, not 100% tested.

D.C. CHARACTERISTICS—PROGRAMMING

$T_A = 25°C \pm 5°C$, $V_{CC} = 6V \pm 0.25V$, $V_{DD} = 12.5V \pm 0.5V$

Symbol	Parameter	Min	Max	Units
V_{DDH}	V_{DD} Program Voltage High Level	12	13	V(1)
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V
V_{PH}	PROG Program Voltage High Level	2.0	5.5	V
V_{PL}	PROG Voltage Low Level	−0.5	0.8	V
V_{EAH}	Input High Voltage for EA	12.0	13.0	V(2)
V_{EAL}	EA Voltage Low Level	−0.5	5.25	V
I_{DD}	V_{DD} High Voltage Supply Current		50.0	mA
I_{EA}	EA High Voltage Supply Current		1.0	mA

NOTES:
1. Voltages over 13V applied to pin V_{DD} will permanently damage the device.
2. V_{EAH} must be applied to EA before V_{DDH} and removed after V_{DDL}.
3. V_{CC} must be applied simultaneously or before V_{DD} and must be removed simultaneously or after V_{DD}.

A.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$, $V_{SS} = 0V$, $V_{CC} = V_{DD} = +5V \pm 10\%$

DBB READ

Symbol	Parameter	Min	Max	Units
t_{AR}	CS, A_0 Setup to RD ↓	0		ns
t_{RA}	CS, A_0 Hold After RD ↑	0		ns
t_{RR}	RD Pulse Width	160		ns
t_{AD}	CS, A_0 to Data Out Delay		130	ns
t_{RD}	RD ↓ to Data Out Delay	0	130	ns
t_{DF}	RD ↑ to Data Float Delay		85	ns

DBB WRITE

Symbol	Parameter	Min	Max	Units
t_{AW}	CS, A_0 Setup to WR ↓	0		ns
t_{WA}	CS, A_0 Hold After WR ↑	0		ns
t_{WW}	WR Pulse Width	160		ns
t_{DW}	Data Setup to WR ↑	130		ns
t_{WD}	Data Hold After WR ↑	0		ns

CLOCK

Symbol	Parameter	Min	Max	Units
t_{CY} (UPI-41)	Cycle Time	2.5	9.20	μs(1)
t_{CYC} (UPI-41)	Clock Period	167	613	ns
t_{CY} (UPI-42)	Cycle Time	1.25	9.20	μs(1)
t_{CYC} (UPI-42)	Clock Period	83.3	613	ns
t_{PWH}	Clock High Time	33		ns
t_{PWL}	Clock Low Time	33		ns
t_R	Clock Rise Time		10	ns
t_F	Clock Fall Time		10	ns

NOTE:
1. $t_{CY} = 15/f(XTAL)$

A.C. CHARACTERISTICS DMA

Symbol	Parameter	Min	Max	Units
t_{ACC}	DACK to WR or RD	0		ns
t_{CAC}	RD or WR to DACK	0		ns
t_{ACD}	DACK to Data Valid	0	130	ns
t_{CRQ}	RD or WR to DRQ Cleared		110	ns(1)

NOTE:
1. $C_L = 150$ pF.

A.C. CHARACTERISTICS—PROGRAMMING

$T_A = 25°C \pm 5°C$, $V_{CC} = 6V \pm 0.25V$, $V_{DDL} = +5V \pm 0.25V$, $V_{DDH} = 12.5V \pm 0.5V$
(8741AH/8742AH ONLY)

Symbol	Parameter	Min	Max	Units
t_{AW}	Address Setup Time to RESET ↑	$4t_{CY}$		
t_{WA}	Address Hold Time After RESET ↑	$4t_{CY}$		
t_{DW}	Data in Setup Time to PROG ↓	$4t_{CY}$		
t_{WD}	Data in Hold Time After PROG ↑	$4t_{CY}$		
t_{PW}	Initial Program Pulse Width	0.95	1.05	ms(1)
t_{TW}	Test 0 Setup Time for Program Mode	$4t_{CY}$		
t_{WT}	Test 0 Hold Time After Program Mode	$4t_{CY}$		
t_{DO}	Test 0 to Data Out Delay		$4t_{CY}$	
t_{WW}	RESET Pulse Width to Latch Address	$4t_{CY}$		
t_r, t_f	PROG Rise and Fall Times	0.5	100	μs
t_{CY}	CPU Operation Cycle Time	2.5	3.75	μs
t_{RE}	RESET Setup Time Before EA ↑	$4t_{CY}$		
t_{OPW}	Overprogram Pulse Width	2.85	78.75	ms(2)
t_{DE}	EA High to V_{DD} High	$1t_{CY}$		

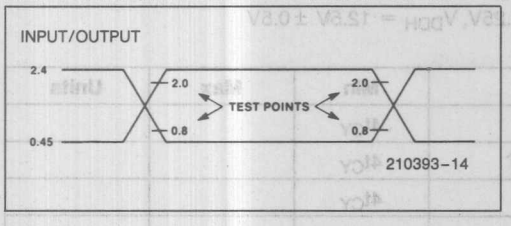
NOTES:
1. Typical Initial Program Pulse width tolerance = 1 ms ±5%.
2. This variation is a function of the iteration counter value, X.
3. If TEST 0 is high, t_{DO} can be triggered by RESET ↑.

A.C. CHARACTERISTICS PORT 2 $T_A = 0°C$ to $+70°C$, $V_{CC} = +5V \pm 10\%$

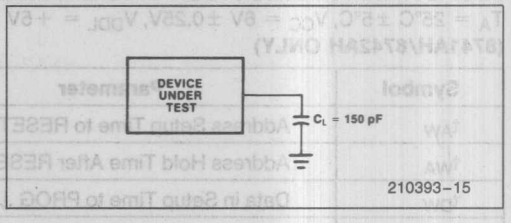
Symbol	Parameter	$f(t_{CY})$(3)	Min	Max	Units
t_{CP}	Port Control Setup Before Falling Edge of PROG	$1/15\ t_{CY} - 28$	55		ns(1)
t_{PC}	Port Control Hold After Falling Edge of PROG	$1/10\ t_{CY}$	125		ns(2)
t_{PR}	PROG to Time P2 Input Must Be Valid	$8/15\ t_{CY} - 16$		650	ns(1)
t_{PF}	Input Data Hold Time		0	150	ns(2)
t_{DP}	Output Data Setup Time	$2/10\ t_{CY}$	250		ns(1)
t_{PD}	Output Data Hold Time	$1/10\ t_{CY} - 80$	45		ns(2)
t_{PP}	PROG Pulse Width	$6/10\ t_{CY}$	750		ns

NOTES:
1. $C_L = 80$ pF.
2. $C_L = 20$ pF.
3. $t_{CY} = 1.25$ μs.

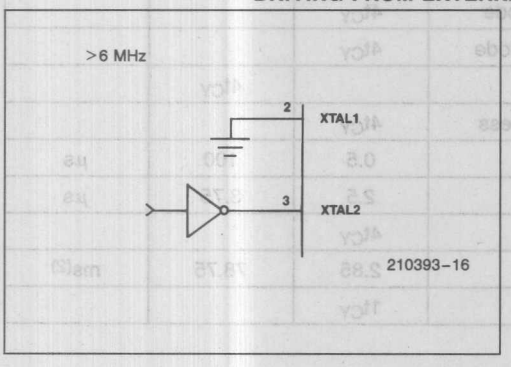
A.C. TESTING INPUT/OUTPUT WAVEFORM

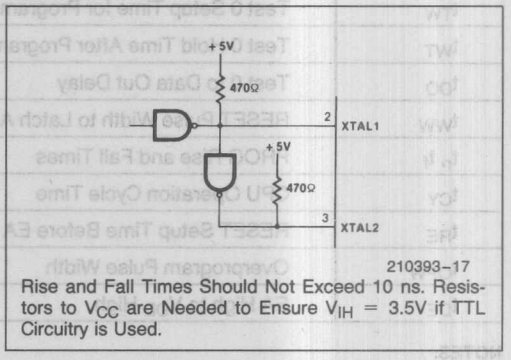


A.C. TESTING LOAD CIRCUIT



DRIVING FROM EXTERNAL SOURCE-TWO OPTIONS

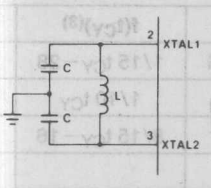




Rise and Fall Times Should Not Exceed 10 ns. Resistors to V_{CC} are Needed to Ensure V_{IH} = 3.5V if TTL Circuitry is Used.

LC OSCILLATOR MODE

L	C	NOMINAL
45 H	20 pF	5.2 MHz
120 H	20 pF	3.2 MHz



$$f = \frac{1}{2\pi\sqrt{LC'}}$$

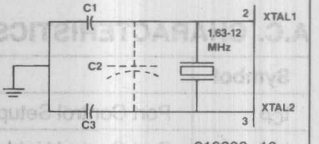
$$C' = \frac{C + 3Cpp}{2}$$

Cpp ≅ 5–10 pF
Pin-to-Pin Capacitance

210393-18

Each C Should be Approximately 20 pF, including Stray Capacitance.

CRYSTAL OSCILLATOR MODE

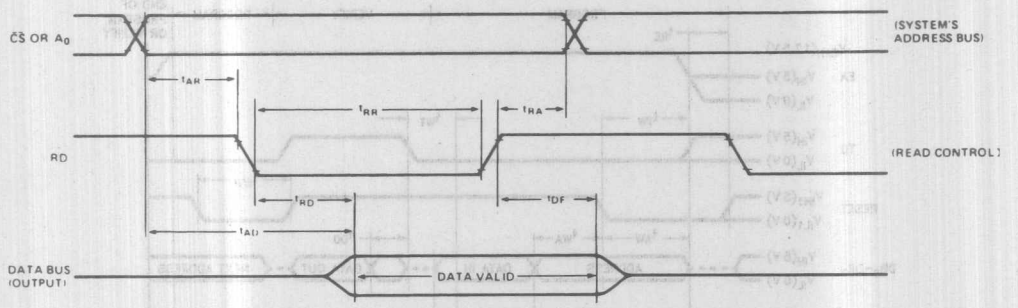


C1 5 pF (STRAY 5 pF)
C2 (CRYSTAL + STRAY) 8 pF
C3 20–30 pF INCLUDING STRAY

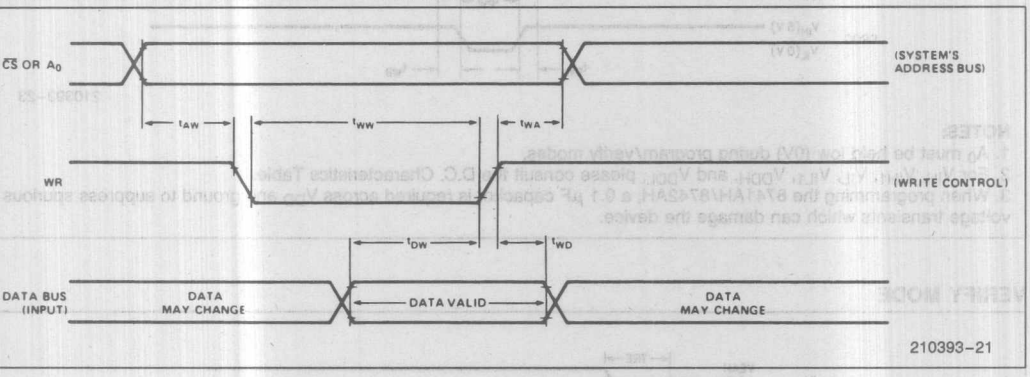
Crystal Series Resistance Should be Less Than 30Ω at 12 MHz; Less Than 75Ω at 6 MHz; Less Than 180Ω at 3.6 MHz.

WAVEFORMS

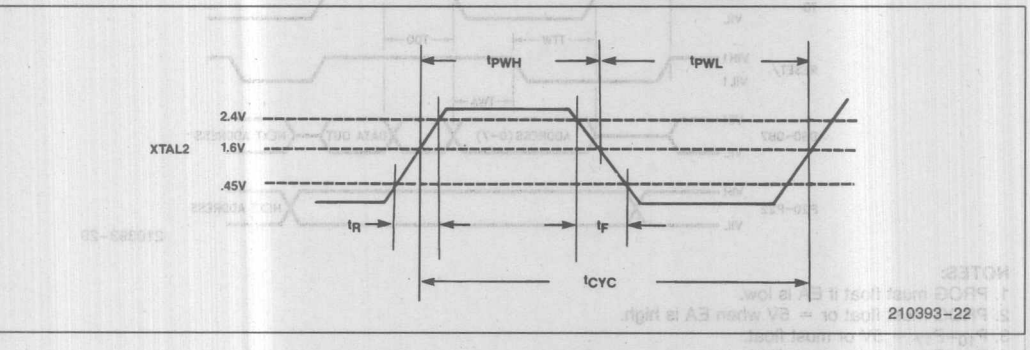
READ OPERATION—DATA BUS BUFFER REGISTER



WRITE OPERATION—DATA BUS BUFFER REGISTER

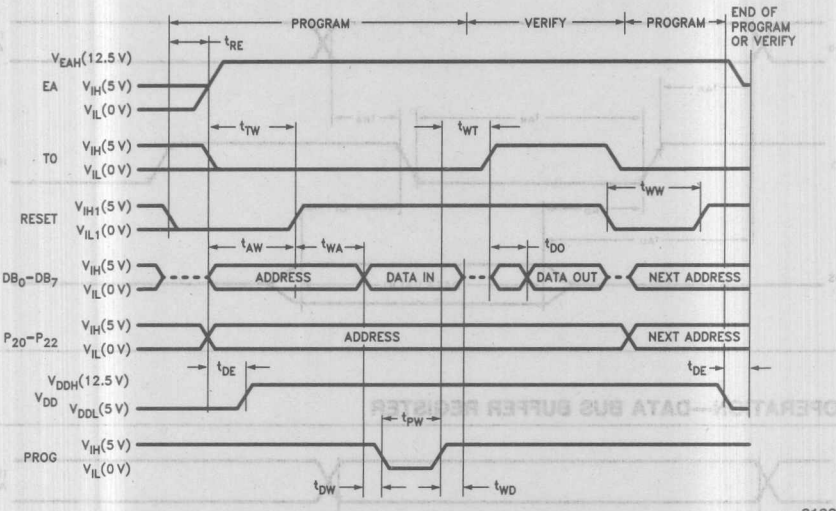


CLOCK TIMING



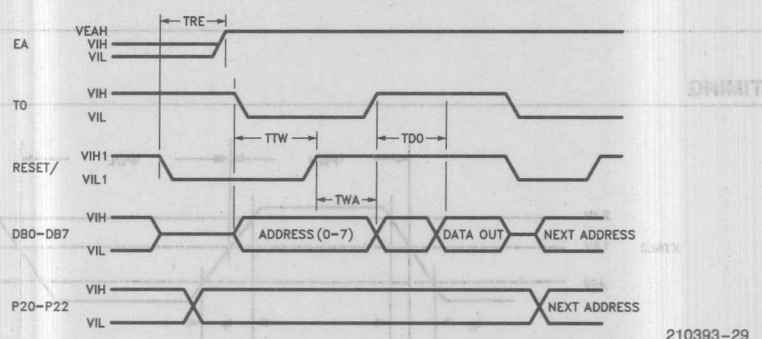
WAVEFORMS (Continued)

COMBINATION PROGRAM/VERIFY MODE



NOTES:
1. A_0 must be held low (0V) during program/verify modes.
2. For V_{IH}, V_{IH1}, V_{IL}, V_{IL1}, V_{DDH}, and V_{DDL}, please consult the D.C. Characteristics Table.
3. When programming the 8741AH/8742AH, a 0.1 μF capacitor is required across V_{DD} and ground to suppress spurious voltage transients which can damage the device.

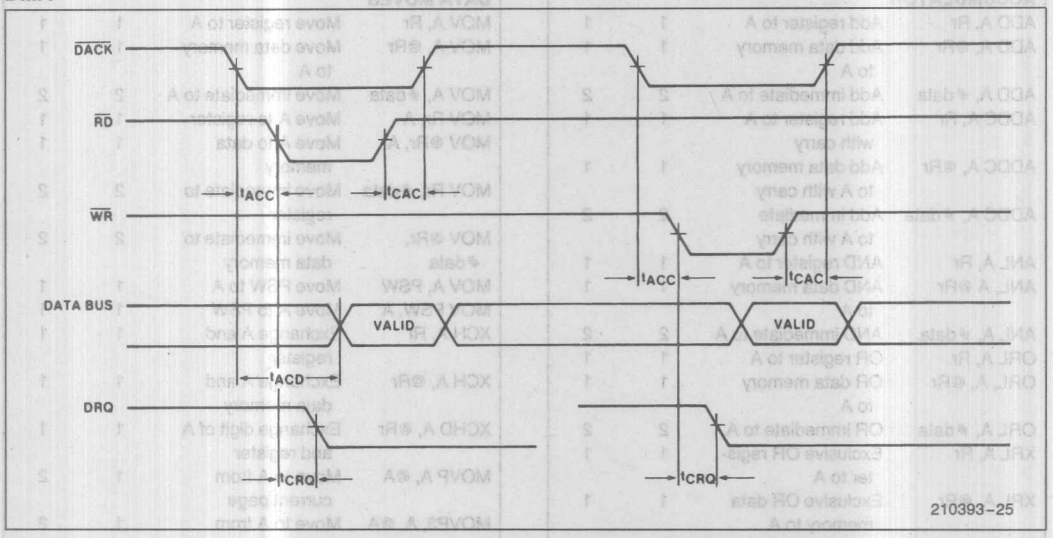
VERIFY MODE



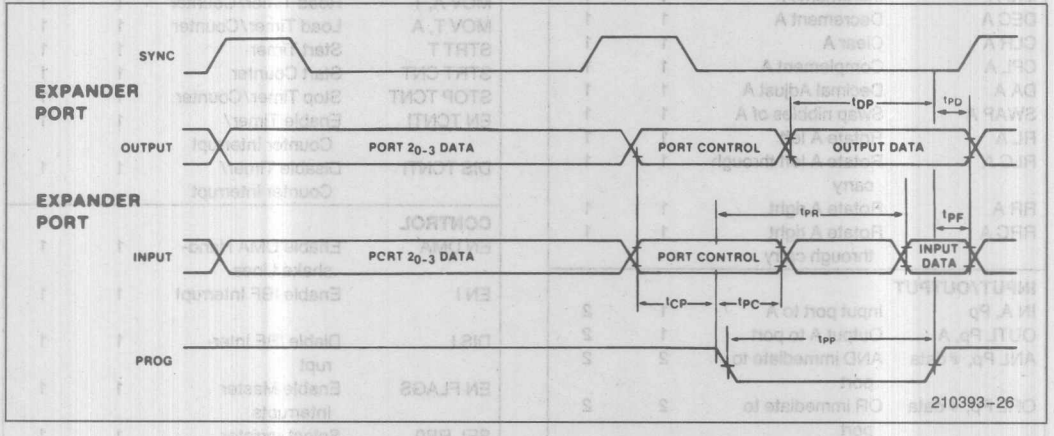
NOTES:
1. PROG must float if EA is low.
2. PROG must float or = 5V when EA is high.
3. P_{10}–P_{17} = 5V or must float.
4. P_{24}–P_{27} = 5V or must float.
5. A_0 must be held low during programming/verify modes.

WAVEFORMS (Continued)

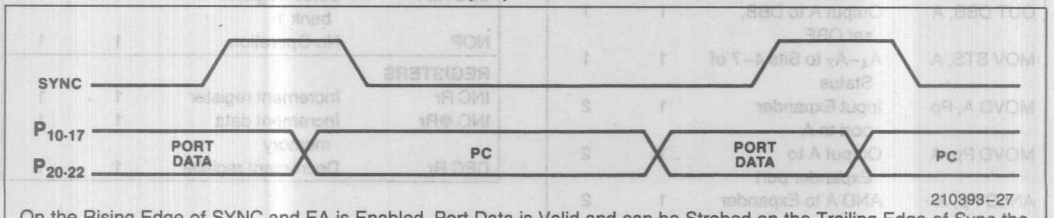
DMA



PORT 2



PORT TIMING DURING EXTERNAL ACCESS (EA)



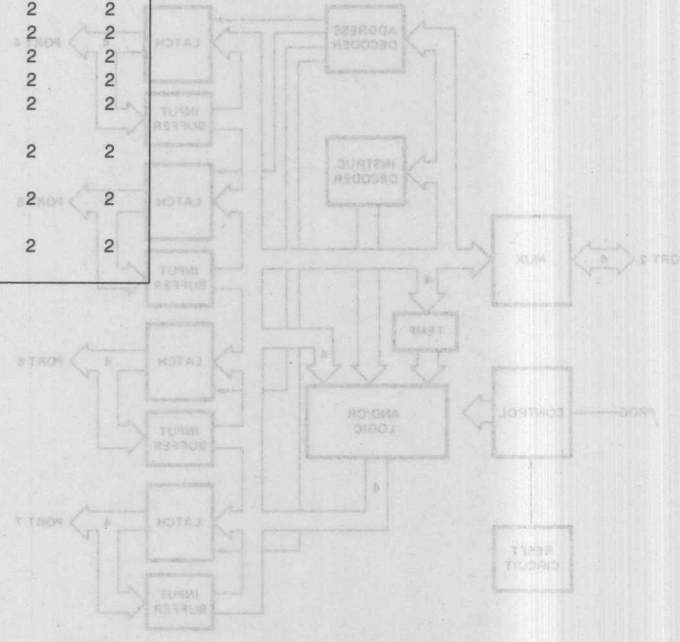
On the Rising Edge of SYNC and EA is Enabled, Port Data is Valid and can be Strobed on the Trailing Edge of Sync the Program Counter Contents are Available.

Mnemonic	Description	Bytes	Cycles
ACCUMULATOR			
ADD A, Rr	Add register to A	1	1
ADD A, @Rr	Add data memory to A	1	1
ADD A, #data	Add immediate to A	2	2
ADDC A, Rr	Add register to A with carry	1	1
ADDC A, @Rr	Add data memory to A with carry	1	1
ADDC A, #data	Add immediate to A with carry	2	2
ANL A, Rr	AND register to A	1	1
ANL, A @Rr	AND data memory to A	1	1
ANL A, #data	AND immediate to A	2	2
ORL A, Rr	OR register to A	1	1
ORL, A, @Rr	OR data memory to A	1	1
ORL A, #data	OR immediate to A	2	2
XRL A, Rr	Exclusive OR register to A	1	1
XRL A, @Rr	Exclusive OR data memory to A	1	1
XRL A, #data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
INPUT/OUTPUT			
IN A, Pp	Input port to A	1	2
OUTL Pp, A	Output A to port	1	2
ANL Pp, #data	AND immediate to port	2	2
ORL Pp, #data	OR immediate to port	2	2
IN A, DBB	Input DBB to A, clear IBF	1	1
OUT DBB, A	Output A to DBB, set OBF	1	1
MOV STS, A	A_4–A_7 to Bits 4–7 of Status	1	1
MOVD A, Pp	Input Expander port to A	1	2
MOVD Pp, A	Output A to Expander port	1	2
ANLD Pp, A	AND A to Expander port	1	2
ORLD Pp, A	OR A to Expander port	1	2

Mnemonic	Description	Bytes	Cycles
DATA MOVES			
MOV A, Rr	Move register to A	1	1
MOV A, @Rr	Move data memory to A	1	1
MOV A, #data	Move immediate to A	2	2
MOV Rr, A	Move A to register	1	1
MOV @Rr, A	Move A to data memory	1	1
MOV Rr, #data	Move immediate to register	2	2
MOV @Rr, #data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, Rr	Exchange A and register	1	1
XCH A, @Rr	Exchange A and data memory	1	1
XCHD A, @Rr	Exchange digit of A and register	1	1
MOVP A, @A	Move to A from current page	1	2
MOVP3, A, @A	Move to A from page 3	1	2
TIMER/COUNTER			
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1
CONTROL			
EN DMA	Enable DMA Handshake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Diable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
REGISTERS			
INC Rr	Increment register	1	1
INC @Rr	Increment data memory	1	1
DEC Rr	Decrement register	1	1

Table 2. UPI™ Instruction Set (Continued)

Mnemonic	Description	Bytes	Cycles
SUBROUTINE			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
FLAGS			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
BRANCH			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ Rr, addr	Decrement register and jump	2	2
JC addr	Jump on Carry = 1	2	2
JNC addr	Jump on Carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 Flag = 1	2	2
JF1 addr	Jump on F1 Flag = 1	2	2
JTF addr	Jump on Timer Flag = 1, Clear Flag	2	2
JNIBF addr	Jump on IBF Flag = 0	2	2
JOBF addr	Jump on OBF Flag = 1	2	2
JBb addr	Jump on Accumula- for Bit	2	2



8243
MCS-48® INPUT/OUTPUT EXPANDER

- **Low Cost**
- **Simple Interface to MCS-48® Microcomputers**
- **Four 4-Bit I/O Ports**
- **AND and OR Directly to Ports**
- **24-Pin DIP**
- **Single 5V Supply**
- **High Output Drive**
- **Direct Extension of Resident 8048 I/O Ports**

The Intel® 8243 is an input/output expander designed specifically to provide a low cost means of I/O expansion for the MCS-48 family of single chip microcomputers. Fabricated in 5 volts NMOS, the 8243 combines low cost, single supply voltage and high drive current capability.

The 8243 consists of four 4-bit bidirectional static I/O ports and one 4-bit port which serves as an interface to the MCS-48 microcomputers. The 4-bit interface requires that only 4 I/O lines of the 8048 be used for I/O expansion, and also allows multiple 8243's to be added to the same bus.

The I/O ports of the 8243 serve as a direct extension of the resident I/O facilities of the MCS-48 microcomputers and are accessed by their own MOV, ANL, and ORL instructions.

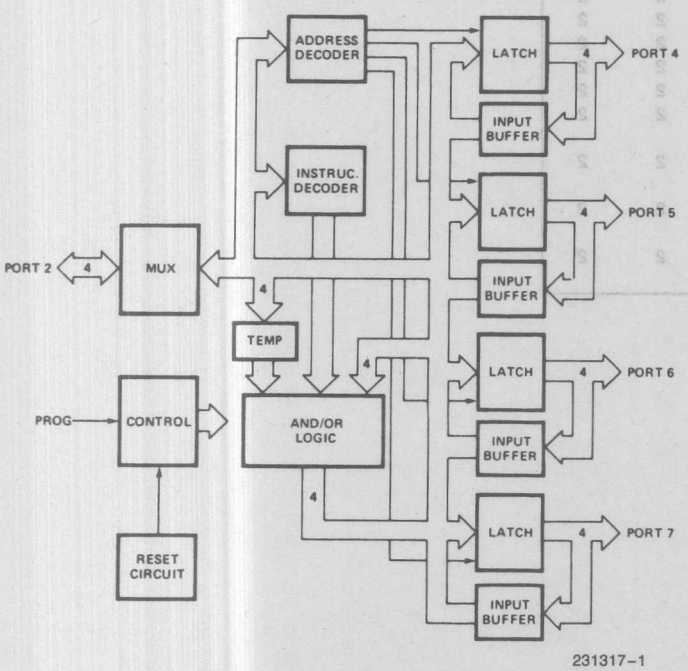


Figure 1. 8243 Block Diagram

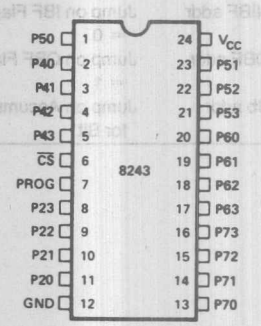


Figure 2. 8243 Pin Configuration

November 1986
Order Number: 231317-001

Table 1. Pin Description

Symbol	Pin No.	Function
PROG	7	**CLOCK INPUT.** A high to low transition on PROG signifies that address and control are available on P20–P23, and a low to high transition signifies that data is available on P20–P23.
$\overline{CS}$	6	**CHIP SELECT INPUT.** A high on CS inhibits any change of output or internal status.
P20–P23	11–8	Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition, contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.
GND	12	0 volt supply.
P40–P43 P50–P53 P60–P63 P70–P73	2–5 1, 23–21 20–17 13–16	Four (4) bit bi-directional I/O ports. May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20–P23 may be directly written, ANDed or ORed with previous data.
V_{CC}	24	+5 volt supply.

FUNCTIONAL DESCRIPTION

General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports 4–7. The following operations may be performed on these ports:

- Transfer Accumulator to Port
- Transfer Port to Accumulator
- AND Accumulator to Port
- OR Accumulator to Port

All communication between the 8048 and the 8243 occurs over Port 2 (P20–P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles.

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if V_{CC} drops below 1V.

Address			Instruction		
P21	P20	Code	P23	P22	Code
0	0	Port 4	0	0	Read
0	1	Port 5	0	1	Write
1	0	Port 6	1	0	ORLD
1	1	Port 7	1	1	ANLD

Write Modes

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG, data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and output. The old data remains latched until new valid outputs are entered.

Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin
With Respect to Ground .....−0.5V to +7V
Power Dissipation 1W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

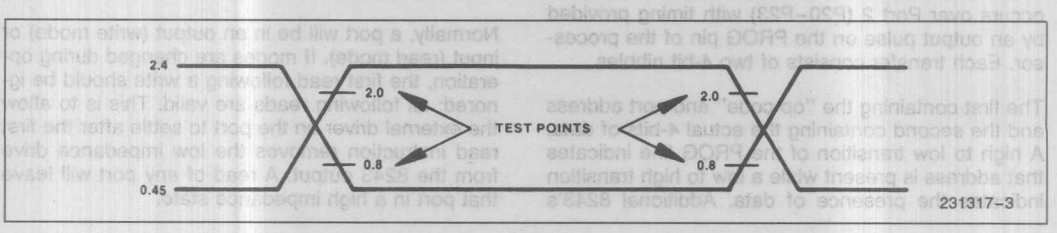
D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC} + 0.5	V	
V_{OL1}	Output Low Voltage Ports 4–7			0.45	V	I_{OL} = 4.5 mA*
V_{OL2}	Output Low Voltage Port 7			1	V	I_{OL} = 20 mA
V_{OH1}	Output High Voltage Ports 4–7	2.4			V	I_{OH} = 240 μA
I_{IL1}	Input Leakage Ports 4–7	−10		20	μA	V_{IN} = V_{CC} to 0V
I_{IL2}	Input Leakage Port 2, CS, PROG	−10		10	μA	V_{IN} = V_{CC} to 0V
V_{OL3}	Output Low Voltage Port 2			0.45	V	I_{OL} = 0.6 mA
I_{CC}	V_{CC} Supply Current		10	20	mA	
V_{OH2}	Output Voltage Port 2	2.4			V	I_{OH} = 100 μA
I_{OL}	Sum of all I_{OL} from 16 Outputs			72	mA	4.5 mA Each Pin

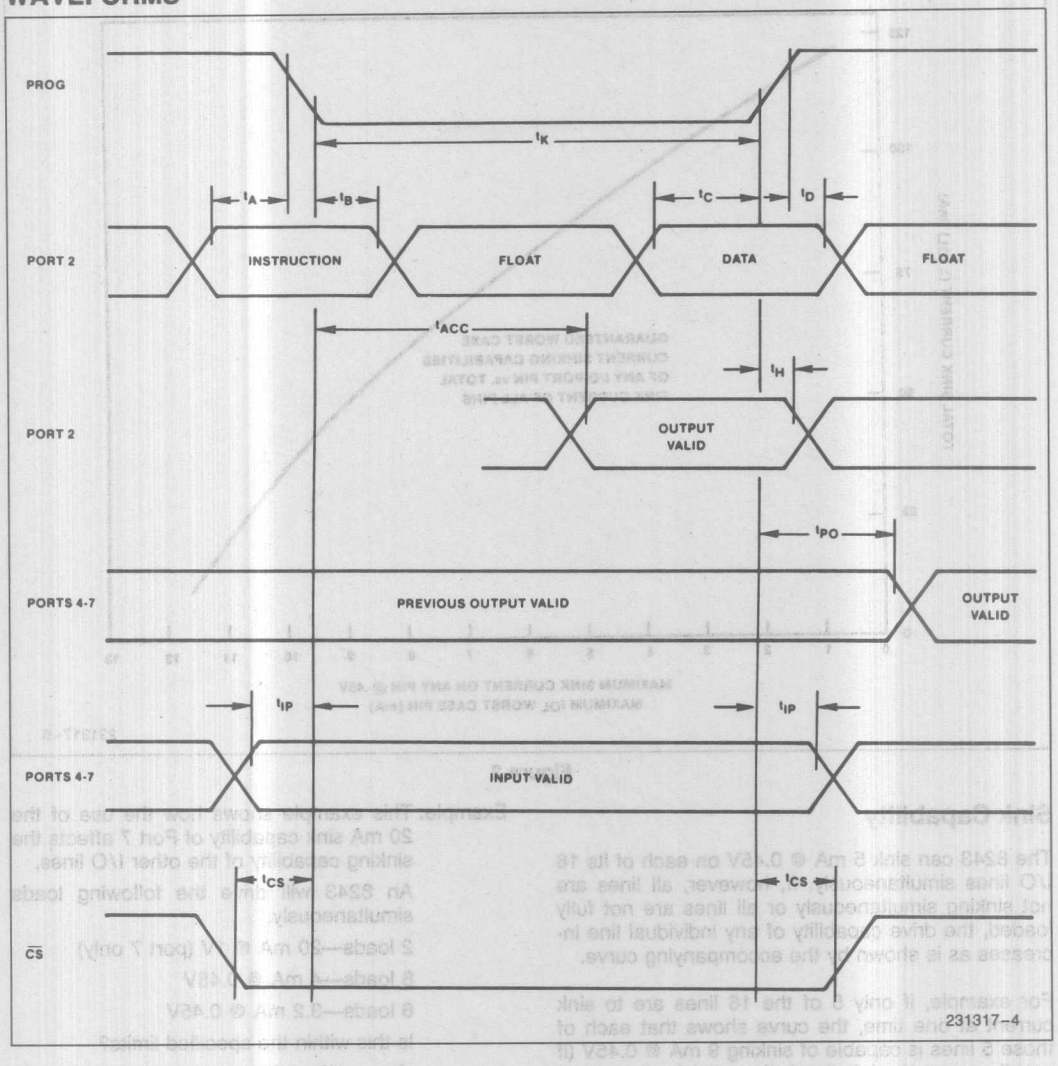
*See following graph for additional sink current capability.

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
t_A	Code Valid Before PROG	100		ns	80 pF Load
t_B	Code Valid After PROG	60		ns	20 pF Load
t_C	Data Valid Before PROG	200		ns	80 pF Load
t_D	Data Valid After PROG	20		ns	20 pF Load
t_H	Floating After PROG	0	150	ns	20 pF Load
t_K	PROG Negative Pulse Width	700		ns	
t_{CS}	CS Valid Before/After PROG	50		ns	
t_{PO}	Ports 4–7 Valid After PROG		700	ns	100 pF Load
t_{LP1}	Ports 4–7 Valid Before/After PROG	100		ns	
t_{ACC}	Port 2 Valid After PROG		650	ns	80 pF Load



WAVEFORMS


PROG
t_K
t_A
t_B
t_C
t_D
PORT 2
INSTRUCTION
FLOAT
DATA
FLOAT
t_{ACC}
t_H
PORT 2
OUTPUT VALID
t_{PO}
PORTS 4-7
PREVIOUS OUTPUT VALID
OUTPUT VALID
t_{IP}
t_{IP}
PORTS 4-7
INPUT VALID
t_{CS}
t_{CS}
$\overline{CS}$
231317-4

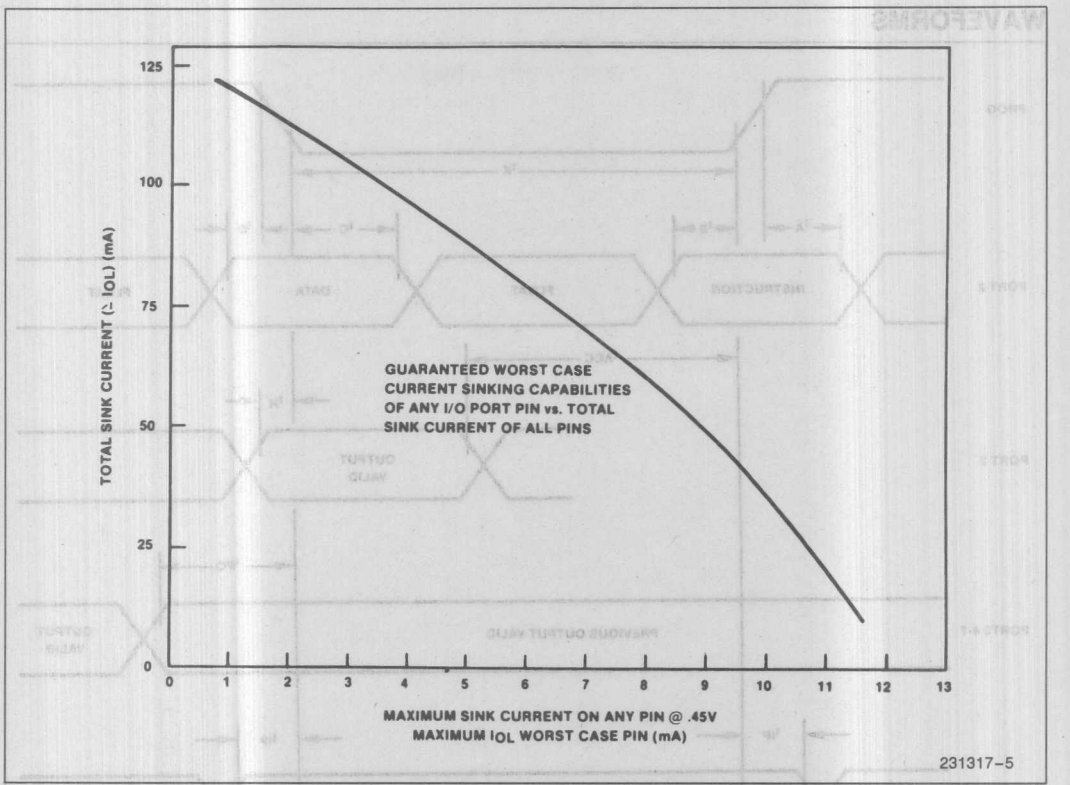


Figure 3

Sink Capability

The 8243 can sink 5 mA @ 0.45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ 0.45V (if any lines are to sink 9 mA the total I_{OL} must not exceed 45 mA or five 9 mA loads).

Example: How may pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

$I_{OL} = 5 \times 1.6\ mA = 8\ mA$

$\epsilon I_{OL} = 60$ mA from curve

pins $= 60\ mA \div 8\ mA/pin = 7.5 = 7$

In this case, 7 lines can sink 8 mA for a total of 56 mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

2 loads—20 mA @ 1V (port 7 only)

8 loads—4 mA @ 0.45V

6 loads—3.2 mA @ 0.45V

Is this within the specified limits?

$\epsilon I_{OL} = (2 \times 20) + (8 \times 4) + (6 \times 3.2) = 91.2$ mA. From the curve: for $I_{OL} = 4$ mA, $\epsilon I_{OL} \approx 93$ mA. Since 91.2 mA $<$ 93 mA the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating ϵI_{OL}, it is the largest current required @ 0.45V which determines the maximum allowable ϵI_{OL}.

NOTE:

A 10 KΩ to 50 KΩ pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

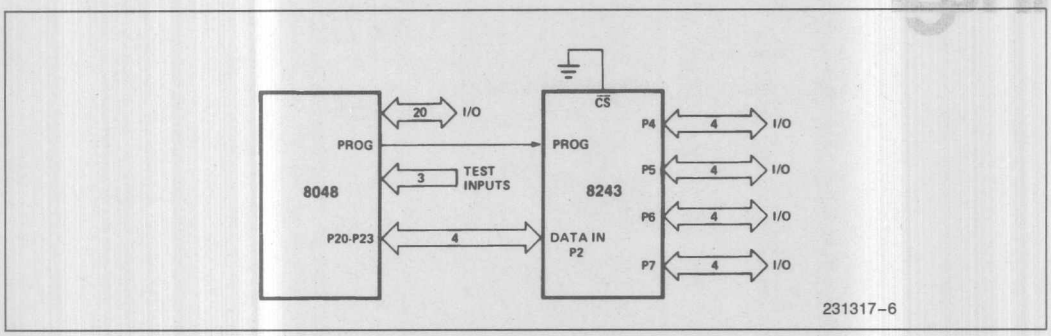


Figure 4. Expander Interface

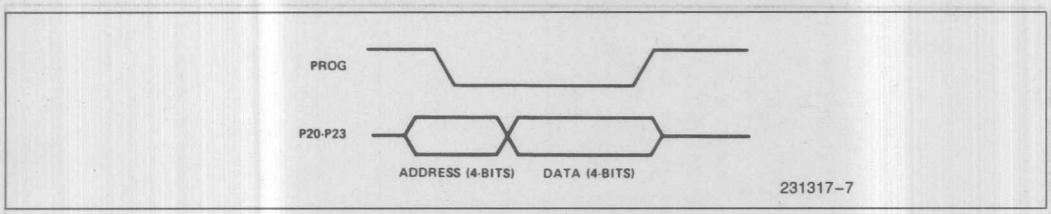


Figure 5. Output Expander Timing

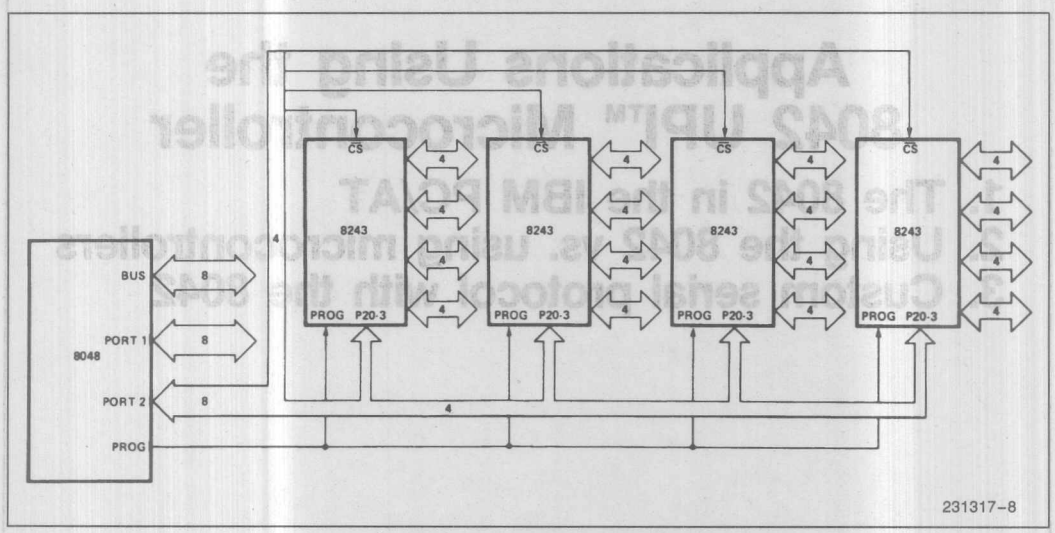
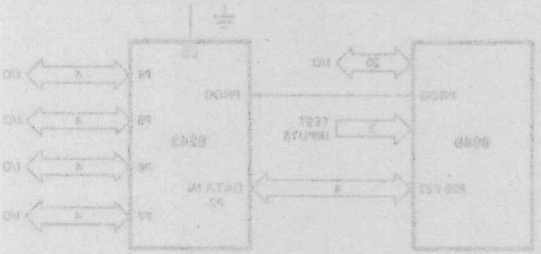


Figure 6. Using Multiple 8243's



June 1985

Applications Using the 8042 UPI™ Microcontroller

1. The 8042 in the IBM PC/AT
2. Using the 8042 vs. using microcontrollers
3. Custom serial protocol with the 8042

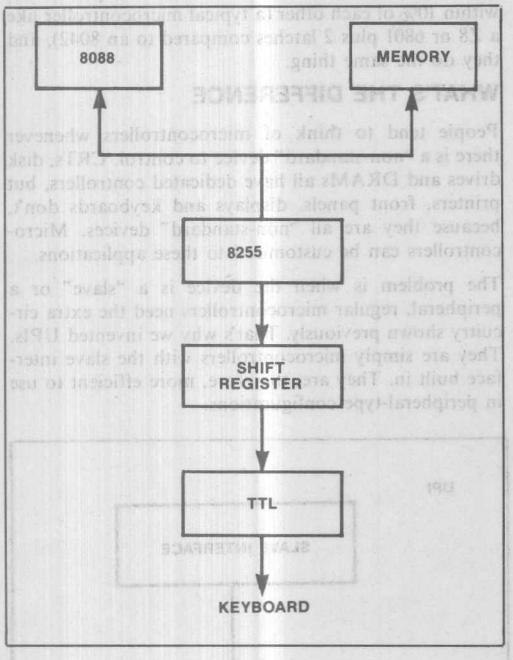
Joe Froelich

Order Number: 231600-001

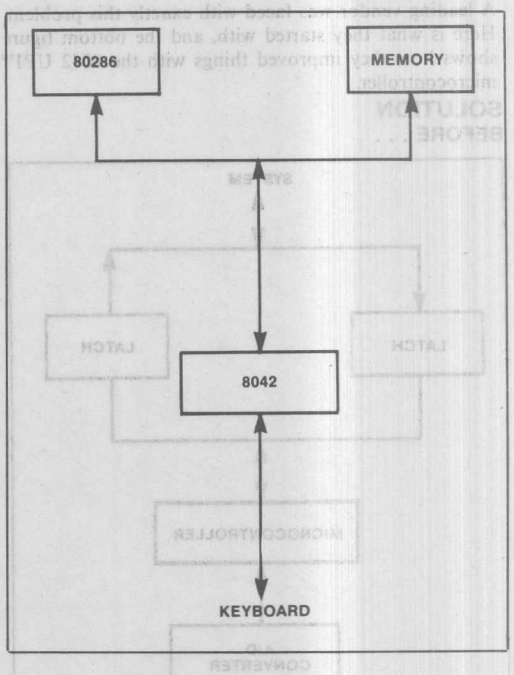
APPLICATION #1
THE 8042 UPI™ MICROCONTROLLER IN THE IBM PC/AT

The following example is an important application of UPIs but there are many more. It is truly a universal device that can be *customized* to all those "non-standard" peripheral control problems. Applications are limited only by imagination. Think UPIs for non-standard peripherals!!

IBM PC/AT (BEFORE) . . .



. . . IBM PC/AT (AFTER)



NEW FUNCTIONS

The 8042 also brings new functions to the PC/AT:

- Keyboard lockup security (front panel key)
- CRT type input to the system
- Diagnostics/self testing of keyboard interface
- Parity check and retry
- PC and PC/AT keyboard interchangeability
- Reset CPU to compatible mode
- Address wrap-around protect in compatible mode

THE FUTURE IS THE KEY

Modifications and upgrades are easy because of the 8042's programmable flexibility and power:

- Change keyboard scan codes (in 8042 ROM)
- Increase functionality of keyboard interface through software and/or unused I/O lines on 8042
- Control other PC/AT functions with these I/O lines

Summary

In short, IBM used the 8042 since it:

- Offloads housekeeping details from the CPU
- Facilitates modular system design
- Offers a customized peripheral
- Provides a clean, efficient upgrade path

These benefits can apply to many of your applications also.

APPLICATION #2
USING THE 8042 VS. USING MICROCONTROLLERS

PROBLEM

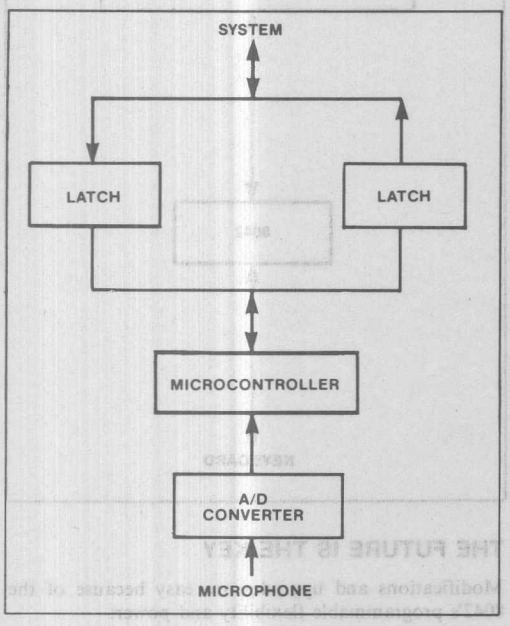
What do you do when you're making SBX and VME modules for a voice digitizing board and you need:

1) an interface to an A/D Converter with
2) 12 MHz operation,
3) an absolute minimum chip count, and
4) very low cost (for the PC market).

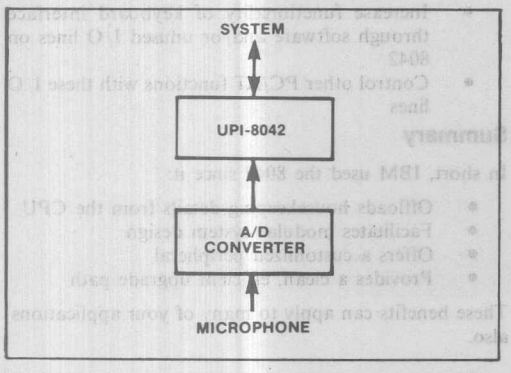
A leading vendor was faced with exactly this problem. Here is what they started with, and the bottom figure shows how they improved things with the 8042 UPI™ microcontroller.

SOLUTION

BEFORE . . .



AFTER . . .



The 8042 integrates two latches and the microcontroller into a single-chip solution.

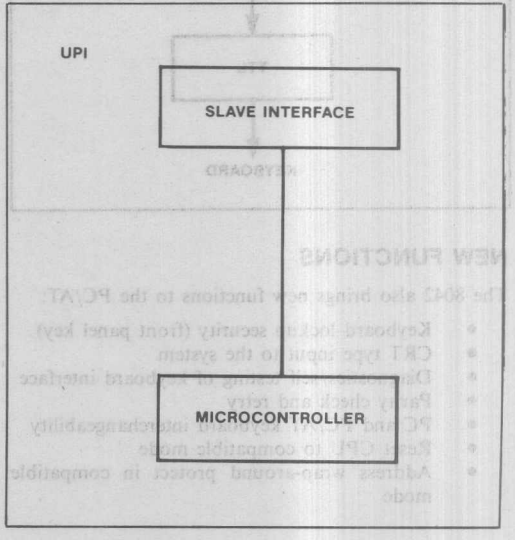
WHY THE SWITCH

After studying the four requirements for this module, it is easy to see why they switched. The first two (A/D interface and 12 MHz) were met by both solutions. However, it is clear the second alternative is much better on board space and on overall cost. There are fewer chips, so they could avoid a multi-layer board and thus save a lot in total cost. Actual chip costs are within 10% of each other (a typical microcontroller like a Z8 or 6801 plus 2 latches compared to an 8042), and they do the same thing.

WHAT'S THE DIFFERENCE

People tend to think of microcontrollers whenever there is a "non-standard" device to control. CRTs, disk drives and DRAMs all have dedicated controllers, but printers, front panels, displays and keyboards don't, because they are all "non-standard" devices. Microcontrollers can be customized to these applications.

The problem is when the device is a "slave" or a peripheral, regular microcontrollers need the extra circuitry shown previously. That's why we invented UPIs. They are simply microcontrollers with the slave interface built in. They are, therefore, more efficient to use in peripheral-type configurations.



UPIs may be misunderstood because of the funny name. They shouldn't be. It's really simple. When faced with non-standard device control, think microcontrollers.

If it's a master-only configuration, think regular microcontrollers. If it's a slave/peripheral configuration, think UPIs.

APPLICATION #3
CUSTOM SERIAL PROTOCOL WITH THE 8042 UPI™ MICROCONTROLLER

BACKGROUND

The 8042 UPI Microcontroller, because of its programmability is being used everywhere, and here is another example. A leading board vendor was designing a communications concentrator board. They wanted a way to:

1) interface 8 serial channels to a minicomputer bus
2) operate these at 4800 baud
3) use one board
4) provide a custom serial protocol that
   - communicated commands and data packets
   - assembled the data packets
   - provided handshaking signals
   - checked for framing, timing, parity, noise, modem and synchronization errors
   - provided self-test diagnostics

THE 8042 SOLUTION

There certainly wasn't an "off-the-shelf" chip that would satisfy the above requirements, and using the main CPU would have caused tremendous system performance degradation. They needed all of these features to offer a competitive product, so they looked to the 8042 UPI Microcontroller. Since the speed requirements were not too great (4800 baud), they could implement the protocol in software. The 8042's programmability gave them all the flexibility needed to incorporate the formatting, handshaking and error checking desired. Moreover, the on-chip slave interface made communication with the minicomputer's bus a snap.

SUMMARY

In short, the 8042 allowed this company to implement a custom serial communication protocol that in turn allowed them to offer a customized, competitive interface board to their customers.

➧**Don't some of your applications need customized interfaces?**

intel

APPLICATION NOTE

AP-161

November 1986

Complex Peripheral Control with the UPI-42AH

CHRISTOPHER SCOTT
TECHNICAL MARKETING ENGINEER

Order Number: 230795-002

INTRODUCTION

The UPI-42AH represents a significant growth in UPI capabilities resulting in a broader spectrum of applications. The UPI-42AH incorporates twice the EPROM/ROM of the UPI-41AH, 2048 vs 1024 bytes, 256 bytes of RAM, and operates at a maximum speed twice that of the UPI-41AH, i.e., 12MHz vs 6MHz. The ROM based 8042 and the EPROM based 8742AH provide more highly integrated solutions for complex stepping motor and dot matrix printer applications. Those applications previously requiring a microprocessor plus a UPI chip can now be implemented entirely with the UPI-42AH.

The software features of the UPI-42AH, such as indirect Data and Program Memory addressing, two independent and selectable 8 byte register banks, and directly software testable I/O pins, greatly simplify the external interface and software flow. The software and hardware design of the UPI-42AH allows a complex peripheral to be controlled with a minimum of external hardware.

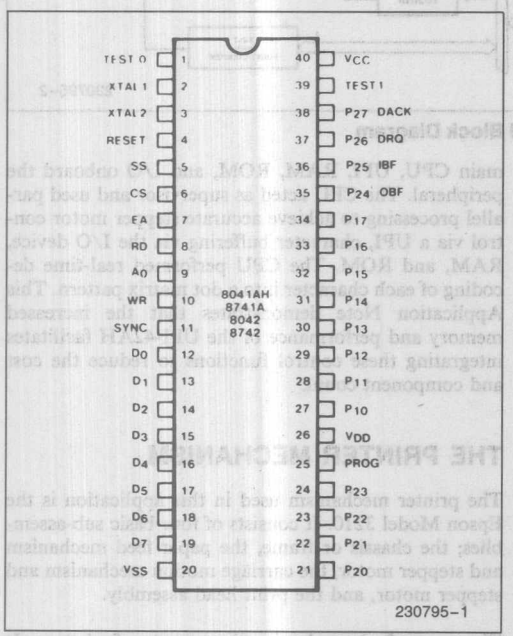


Figure 1. UPI-42AH Pin Configuration

Many microcomputer systems need real time control of peripheral devices such as a printer, keyboard, complex motor control or process control. These medium speed but still time consuming tasks require a fair amount of system software overhead. This processing burden can be reduced by using a dedicated peripheral control processor.

Until recently, the dedicated control processor approach was usually not cost effective due to the large number of components needed; CPU, RAM, ROM, I/O, and Timer/Counters. To help make the approach more cost effective, in 1977 Intel introduced the UPI-41 family of Universal Peripheral Interface controllers consisting of an 8041AH (ROM) device and an 8741AH (EPROM) device. These devices integrated the common microprocessor system functions into one 40 pin package. The UPI-42AH family, consisting of the 8042 and 8742AH, further extends the UPI's cost effectiveness through more memory and higher speed.

Another member of the UPI family is the Intel 8243 Input/Output Expander chip. This chip provides the UPI-41AH and UPI-42AH with up to 16 additional independently programmable I/O lines, and interfaces directly to the UPI-41AH/42AH. Up to seven 8243s can be cascaded to provide over 100 I/O lines.

The UPI is a single chip microcomputer with a standard microprocessor interface. The UPI's architecture, illustrated in Figure 3, features on-chip program memory, ROM (8041A/8042) or EPROM (8741A/8742AH), data memory (RAM), CPU, timer/counter, and I/O. Special interface registers are provided which enable the UPI to function as a peripheral to an 8-bit central processor.

Using one of the UPI devices, the designer simply codes his proprietary peripheral control algorithm into the UPI device itself, rather than into the main system software. The UPI device then performs the peripheral control task while the host processor simply issues commands and transfers data. With the proliferation of microcomputer systems, the use of UPIs or slave microprocessors to off load the main system microprocessor has become quite common.

This Application Note describes how the UPI-42AH can be used to control dot matrix printing and the printer mechanism, using stepper motors for carriage/print head assembly and paper feed motion. Previous Intel Application Notes AP-27, AP-54, and AP-91 describe using intelligent processors and peripherals to control single solenoid driven printer mechanisms with 80 character line buffering and bidirectional printing. This Application Note expands on these previous themes and extends the concept of complex device control by incorporating full 80 character line buffering, bidirectional printing, as well as drive and feedback control of two four phase stepper motors.

The Application Note assumes that the reader is familiar with the 8042/8742AH and 8243 Data Sheets, and UPI-41AH/42AH Assembly Language. Although some background information is included, it also assumes a basic understanding of stepper motors and dot matrix printer mechanisms. A complete software listing is included in Appendix A.

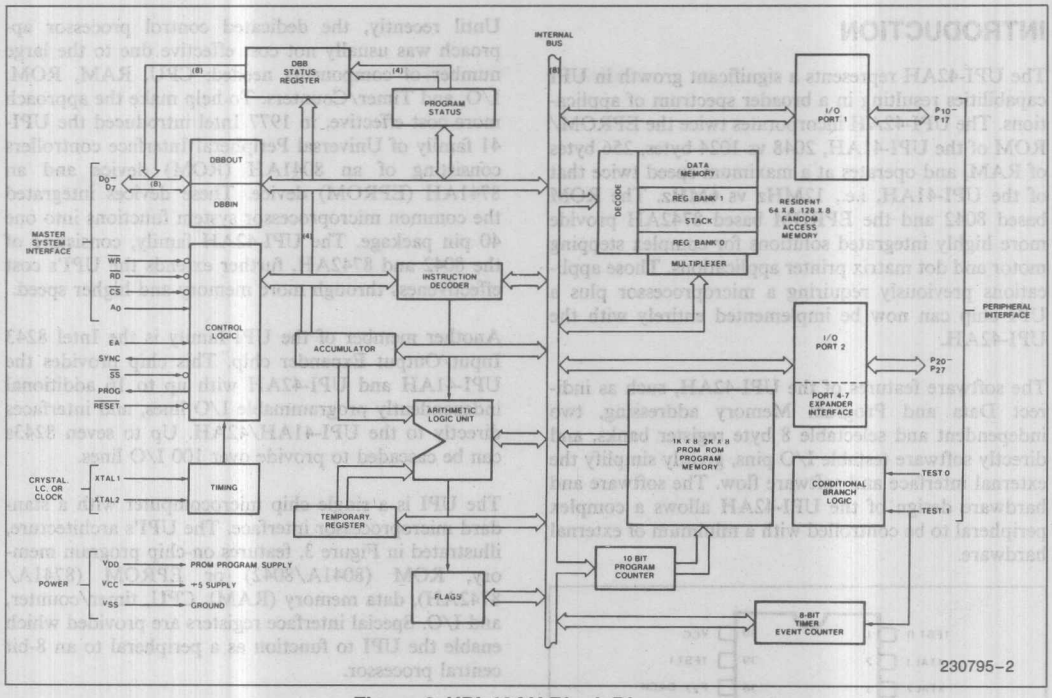


Figure 2. UPI-42AH Block Diagram

DOT MATRIX PRINTING

A dot matrix printer print head typically consists of seven to nine solenoids, each of which drives a stiff wire, or hammer, to impact the paper through an inked ribbon. Characters are formed by firing the solenoids to form a matrix of "dots" (impacts of the wires). Figure 4 shows how the character "E" is formed using a 5 × 7 matrix. The columns are labeled C1 through C5, and the rows R1 through R7. The print head moves left-to-right across the paper, so that at time T1 the head is over column C1. The character is formed by activating the proper solenoid as the print head sweeps across the character position.

Dot matrix printers are a cost effective way of providing good quality hard copy output for microcomputer systems. There is an ever increasing demand for the moderately priced printer to provide more functionality with improved cost and performance. Using stepper motors to control the paper feed and carriage/print head assembly motion is one way of enabling the dot matrix printer to provide more capabilities, such as expanded or contracted characters, dot or line graphics, variable line and character spacing, and subscript or superscript printing.

However, stepper motors require fairly complex control algorithms. Previous solutions involved the use of a main CPU, UPI, RAM, ROM, and I/O onboard the peripheral. The CPU acted as supervisor and used parallel processing to achieve accurate stepper motor control via a UPI, character buffering via the I/O device, RAM, and ROM. The CPU performed real-time decoding of each character into a dot matrix pattern. This Application Note demonstrates that the increased memory and performance of the UPI-42AH facilitates integrating these control functions to reduce the cost and component count.

THE PRINTER MECHANISM

The printer mechanism used in this application is the Epson Model 3210. It consists of four basic sub-assemblies; the chassis or frame, the paper feed mechanism and stepper motor, the carriage motion mechanism and stepper motor, and the print head assembly.

The paper feed mechanism is a tractor feed type. It accommodates up to 8.5 inch wide paper (not including tractor feed portion). There is no platen as such; the paper is moved through the paper guide by two sprocketed wheels mounted on a center sprocket shaft. The sprocket shaft is driven by a four phase stepper motor. The rotation of the stepper motor is transmitted to the sprocket shaft through a series of four reduction gears.

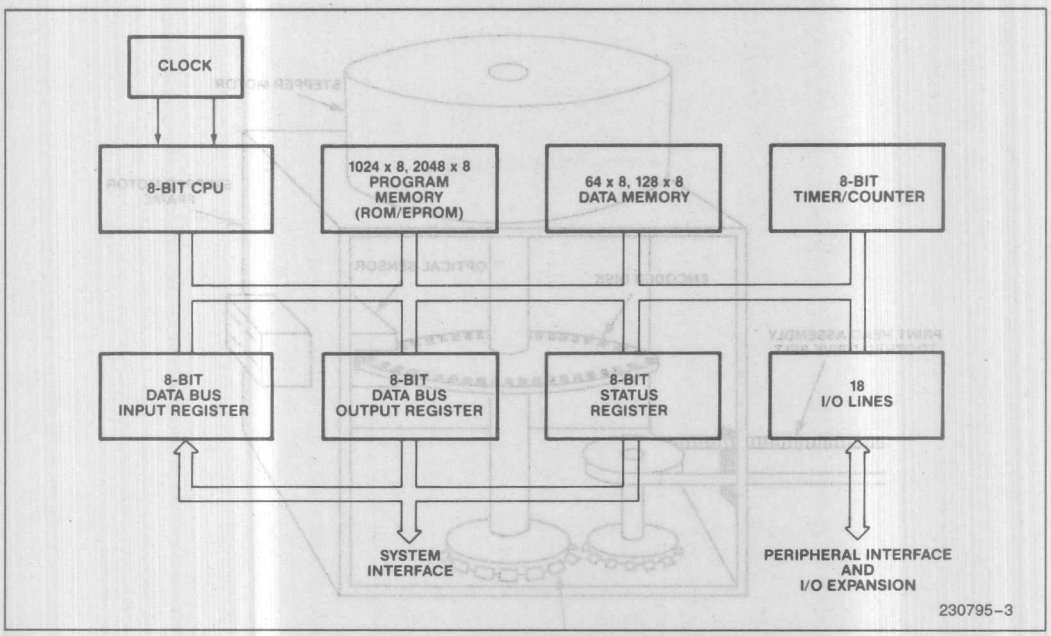


Figure 3. UPI-41AH, 42AH Functional Block Diagram

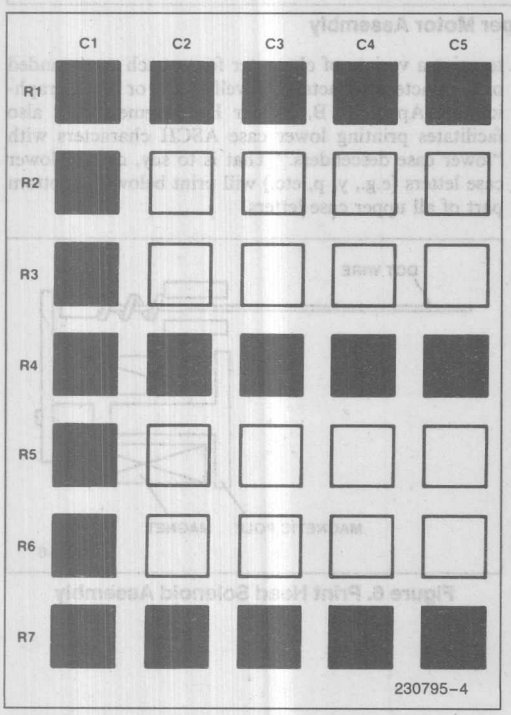


Figure 4. Character E in 5 x 7 Dot Matrix Format

The carriage motion mechanism consists of another four phase stepper motor which controls the left-to-right or right-to-left print head assembly motion. The print speed is 80 CPS maximum. Both the speed of the stepper motor and the movement of the print head assembly are independently controllable in either direction. The rotation of the stepper motor is converted to the linear motion of the print head assembly via a series of reduction gears and a toothed drive belt. The drive belt also controls a second set of reduction gears which advances the print ribbon as the print head assembly moves.

Two optical sensors provide feedback information on the carriage assembly position and speed. The first of these optical sensors, called the 'HOME RESET' or HR, is mounted near the left-most physical position to which the print head assembly can move. As the print head assembly approaches the left-most position, a flange on the print head assembly interferes with the light source and sensor, causing the output of the sensor to shift from a logic level one to zero. The right-most printer position is monitored in software rather than by another optical sensor. The right-most print position is a function of the number of characters printed and the distance required to print them.

The second optical sensor, called the 'PRINT TIMING SIGNAL' or PTS, provides feedback on carriage stepper motor velocity and relative position within a

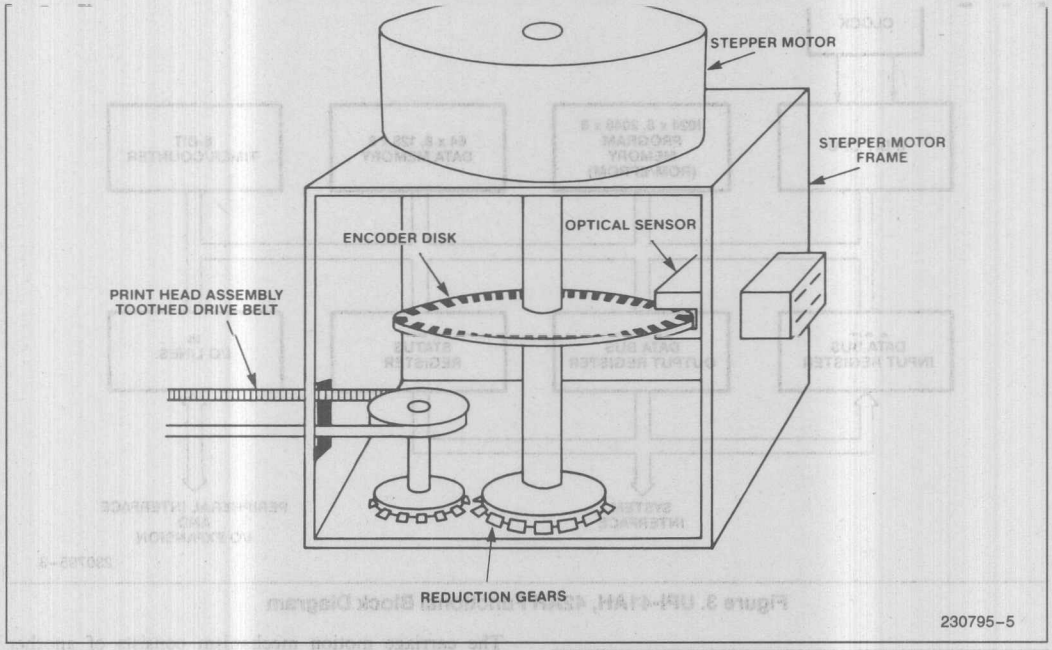


Figure 5. Carriage Stepper Motor Assembly

given step of the motor. The feedback is generated by the optical sensor as an "encoder disk" moves across it. Figure 5 illustrates the carriage stepper motor, optical sensor, encoder disk and reduction gears, and drive belt assembly. The optical sensor outputs a pulse train with the same period as the phase shift signal used to drive the stepper, but slightly out of phase with it when the motor is at a constant speed (see Software Functional Block: Phase Shift Data for additional details). The disk acts as a timing wheel, providing feedback to the UPI software of the carriage speed, position, and optimum position for energizing the print head solenoids. The two optical sensors are monitored under software and provide the critical feedback needed to control the print head assembly and paper feed motion accurately. The process of stepper motor drive and control via feedback signals is called "closed loop" stepper motor control, and is covered in more detail in the software discussion.

The print head assembly consists of nine solenoids and nine wires or hammers. Figure 6 illustrates a print head assembly. The available dot matrix measures 9 x 9. This large matrix enables the Epson 3210 print mechanism to print a variety of character fonts, such as expanded or contracted characters, as well as line or block graphics (see Appendix B, Printer Enhancements). It also facilitates printing lower case ASCII characters with "lower case descenders." That is to say, certain lower case letters (e.g., y, p, etc.) will print below the bottom part of all upper case letters.

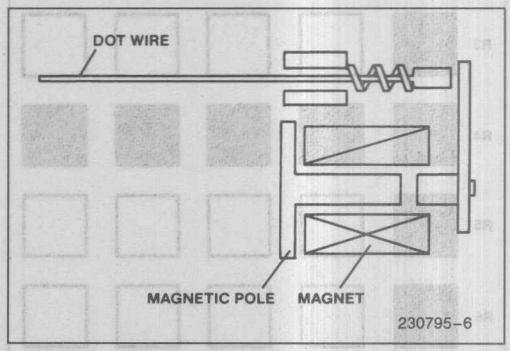


Figure 6. Print Head Solenoid Assembly

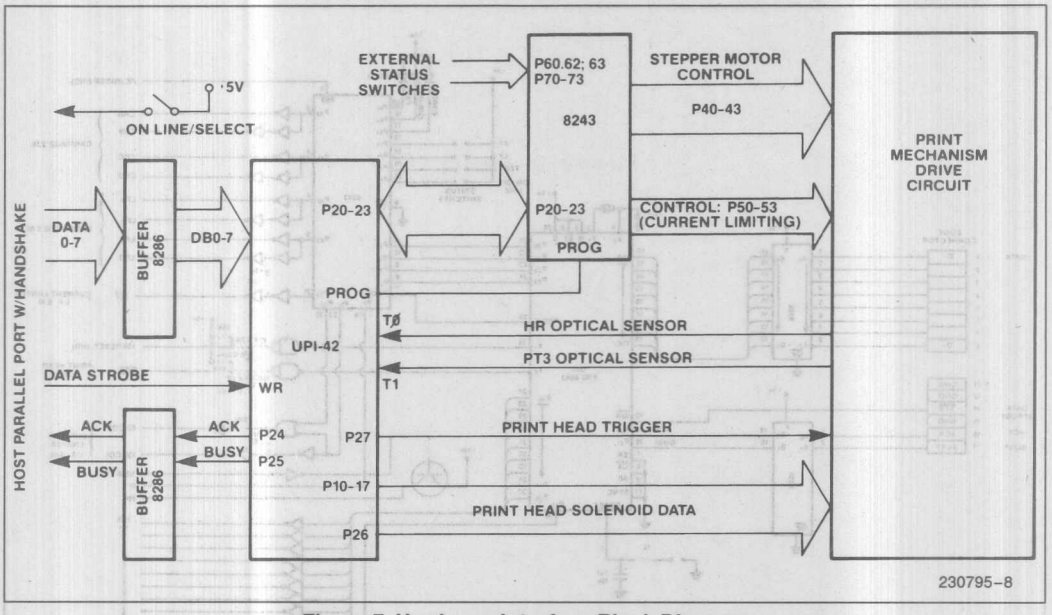


Figure 7. Hardware Interface Block Diagram

HARDWARE DESCRIPTION

Figure 7 shows a block diagram of the UPI-42AH and 8243 interface to the printer mechanism drive circuit. A complete schematic is shown in Figure 8. The UPI-42AH provides all signals necessary to control character buffering and handshaking, paperfeed and carriage motion stepper motor timing, print head solenoid activation, and monitoring of external status switches.

The Epson 3210 printer mechanism manual recommends a specific interface circuit to provide proper drive levels to the stepper motors windings and print head solenoids. The hardware interface used for this Application Note followed those recommendations exactly (see Appendix C, Printer Mechanism Drive Circuit Schematics).

I/O Ports

The lower half of the UPI-42AH Port 2, pins 0–3, provides an interface to the 82431 I/O expander. The PROG pin of the UPI-42AH is used as a strobe to clock address and data information via the Port 2 interface. The extra 16 I/O lines of the 8243 become PORTS 4, 5, 6, and 7 to the UPI software. Combined, the UPI-42AH and 8243 provide a total of 28 independently programmable I/O line. These lines are used as shown in Figure 9.

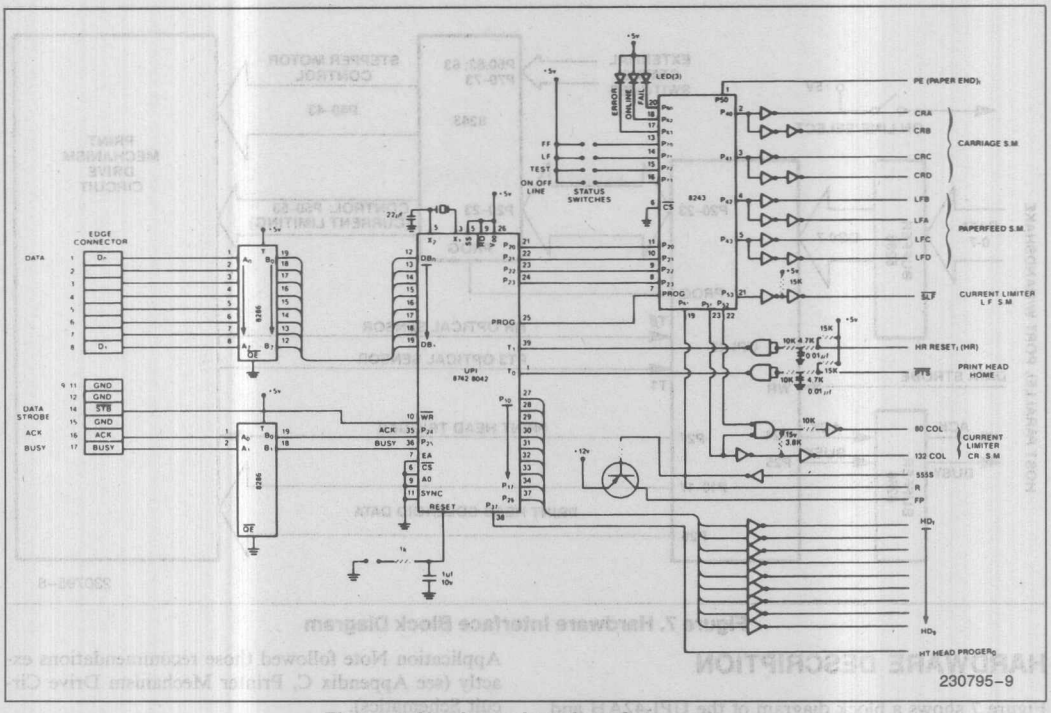


Figure 8. Hardware Interface Schematic

Port	No. of lines	Bits	I/O	Description
1	8	0–7	O	Character dot column data to print head solenoids
2	1	6	O	(same)
2	1	7	O	Print head solenoid trigger
2	2	4, 5	O	Host system data transfer handshaking (ACK/BUSY)
4	4	0–3	O	Carriage & paper feed stepper motors
5	3	1–3	O	Stepper motor select and current limiting
5	1	0	I	Paper End sense
6	1	1	O	Print head trigger reset
6	3	0, 2, 3	-	(unused)
7	5	0–3	I	External status switches; (LF, FF, TEST, ON/OFF Line)

Figure 9. UPI-42AH and 8243 I/O Port Map

NOTE:
The notation used in the balance of this Application Note, when referring to a port number and a particular pin or bit, is Port 23 rather than Port 2 bit 3.

The two printer mechanism optical sensors, discussed in the Printer Mechanism description are tied to the two "Test Input" pins, T0 and T1, of the UPI-42AH through a buffer circuit for noise suppression. These inputs are directly testable in software.

Host System Interface

The host system interfaces to the printer through a parallel port to the UPI-42AH Data Bus. Four handshaking signals are used to control data transfer; Data Strobe (STB/), Acknowledge (ACK), Busy (BUSY), and Online or Select. The Data Strobe line of the host parallel port is tied directly to the UPI-42AH WR/ pin. This provides a low going pulse on the UPI-42AH WR/ pin whenever a data byte is written to the UPI-42AH. The ACK and BUSY handshake signals are tied to two UPI-42AH I/O port lines for software control of data transfer. The "On Line" handshake signal is tied to a single-pole single-throw fixed position switch, which externally enables or disables character transfer from the host system. Characters transmitted to the UPI-42AH by the host are loaded into the UPI-42AH Data Bus Buffer In (DBBIN) register, and the Input Buffer Full (IBF) interrupt and UPI-42AH status flag are set (see Figure 9. UPI-42AH and 8243 I/O Ports).

Stepper Motor Interface

Port 4 (41-43) of the 8243, provides both carriage and paper feed stepper motor phase shift signals to the printer mechanism drive circuit. Each of the two stepper motors is driven by 2 two phase excitation signals (4 phases). Figure 10 shows the wave form for each stepper motor. Each signal consists of two components (Sig. 1 A/B & Sig. 2 C/D) 180 degrees out of phase with the other. Each of these signal pairs (A/B & C/D) is 90 degrees out of phase with the other pair. For each signal pair, one port line supplies both halves by using an inverter.

Each of the resulting eight stepper motor drive signals is interfaced to a discrete drive transistor through an inverter. The emitter of the drive transistor is tied to the open collector of the inverter to provide high current sinking capability for the drive transistor. Each half of the motor winding is tied to the collector of the drive transistor (see Appendix C, Printer Mechanism Drive Circuit Schematic).

Each stepper motor requires two current levels for operation. These levels are called "Rush" current and "Hold" current. Rush current refers to the high current required to cause the rotor to rotate within its windings as the polarity of the power applied to the windings is changing. Each change in the polarity of the power applied to the motor windings is called a step of phase shift. Hold current refers to the low level of current required to stabilize and maintain the rotor in a fixed position when the polarity is applied to the windings is not changing. Hold current is simply Rush current with a current limiting transistor switched in. Switching from Hold to Rush current "selects" or enables that

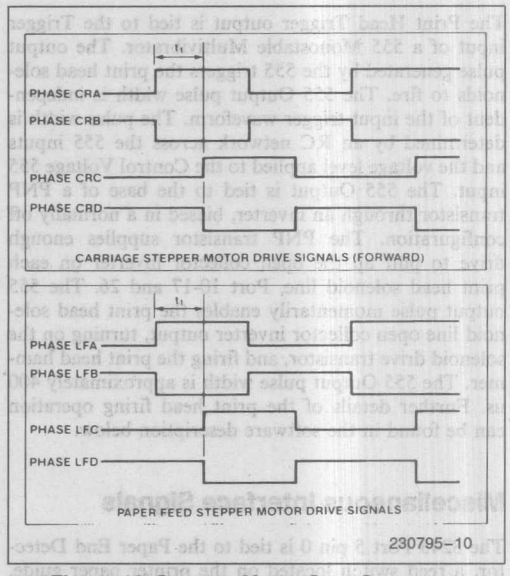


Figure 10. Stepper Motor Step Sequence Waveforms

stepper motor to move with the next step signal output. In the balance of this Application Note, the term "select" will be used to refer to turning on Rush current, and "deselect" will refer to switching to Hold current.

Three 8243 port lines are dedicated to the select/deselect control of the two stepper motors. One line is for the paper feed stepper motor, and two lines are for the carriage motion stepper motor (80 and 132 column). These lines are labeled SLF, 80Col, and 132Col, and are 8243 PORT 53, 52, and 51, respectively.

By varying the voltage applied to the stepper motor biasing circuit and the current, it is possible to vary the distance the motor moves the print head assembly with each step. Enabling one of two different voltage biasing levels, and changing the timing rate at which the motor is stepped, facilitates either 80 or 132 character column printing. Only 80 character column printing is implemented in the software design. Appendix B, Printer Enhancements, details the software algorithm for handling 132 character printing.

Print Head Interface

A total of eleven I/O lines are used to control the print head solenoids and solenoid firing (see Figure 9). Nine are used for character dot data, one for the Print Head Trigger, and one for Reset of the Print Head Trigger circuit. Each of the nine character dot data lines is buffered by an open collector hex inverter.

input of a 555 Monostable Multivibrator. The output pulse generated by the 555 triggers the print head solenoids to fire. The 555 Output pulse width is independent of the input trigger waveform. The pulse width is determined by an RC network across the 555 inputs and the voltage level applied to the Control Voltage 555 input. The 555 Output is tied to the base of a PNP transistor through an inverter, biased in a normally off configuration. The PNP transistor supplies enough drive to pull up the open collector inverter on each print head solenoid line, Port 10-17 and 26. The 555 output pulse momentarily enables the print head solenoid line open collector inverter output, turning on the solenoid drive transistor, and firing the print head hammer. The 555 Output pulse width is approximately 400 us. Further details of the print head firing operation can be found in the software description below.

Miscellaneous Interface Signals

The 8243 Port 5 pin 0 is tied to the Paper End Detector, a reed switch located on the printer paper guide. This sensor detects when the paper is nearly exhausted.

Three LED status lights complete the hardware interface design. One status light is used for each of the following: Power ON/OFF, On/Off Line, and Out of Paper.

BACKGROUND

Before a detailed discussion of the software begins, a few terms and software functions reference throughout the software need introduction.

A. What is a Stepper Motor?

A stepper motor has the ability to rotate in either direction as well as start and stop at predetermined angular positions. The stepper motor's shaft (rotor) moves in precise angular increments for each input step. The displacement is repeated for each input step command, accurately positioning the rotor for a given number and sequence of steps.

The stepper motor controls position, velocity, and direction. The accuracy of stepper motors is generally 5 percent of one step. The number of steps in each revolution of the shaft varies, depending on the intended application.

B. Open/Closed Loop Stepper Motor Drive and Control

The carriage stepper motor is closed loop driven. The paper feed stepper motor is open loop driven. known by the broad headings of open and closed loop. Open loop is simply continuous pulses to drive the motor at a predetermined rate based on the voltage, current, and the timing of the step pulses applied. Closed loop control is characterized by continuous monitoring of the stepper motor, through feedback signals, and adjusting the motor's operation based upon the feedback received.

C. Stepper Motor Drive Phase Shift of Step Sequence

Each change in the polarity of the power applied to the motor windings is called a step or phase shift. The sequence of the steps or phase shifts, and the pattern of polarity changes output to the stepper motor, determines the direction of rotation.

Figure 10 shows the waveforms for each of the two stepper motors. Figure 11 lists the step sequence for carriage motor clockwise rotation, whch moves the print head assembly Left-to-Right. Figure 11 also lists the step sequence for counterclockwise rotations; the print head assembly moves Right-to-Left. Figure 12 lists the step sequence for the paper feed stepper motor clockwise drive. The phase sequence, for either stepper motor, may begin at any point within the sequence list, but must then continue in order.

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	Off	On
2	On	Off	On	Off
3	Off	On	On	Off
4	Off	On	Off	On

Carriage stepper motor rotates clockwise Print head assembly moves from left to right

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	On	Off
2	On	Off	Off	On
3	Off	On	Off	On
4	Off	On	On	Off

Carriage stepper motor rotates counter clockwise Print head assembly moves from right to left

Figure 11. Carriage Stepper Motor Step Sequence

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	On	Off
2	On	Off	Off	On
3	Off	On	Off	On
4	Off	On	On	Off

Figure 12. Paper Feed Stepper Motor Step Sequence

D. Acceleration and Deceleration of Stepper Motors

The carriage stepper motor starts from a fixed position, accelerates to a constant speed, maintains constant speed, and then decelerates to a fixed position. Printing may occur from the time and position the print head assembly reaches constant speed, until the time and position the print head assembly begins to decelerate from constant speed. Whether printing occurs during any carriage stepper motor drive sequence is controlled by software. Figure 18, below, illustrates these components of print head assembly line motion.

Due to inertia, a finite time interval and angular displacement is required to accelerate a stepper motor to its full speed. Conversely, deceleration must begin some time before the final angular position. The time interval and angular displacement of the carriage stepper motor translates into the distance the print head assembly travels before it reaches a constant speed. The distance traveled during acceleration is constant. The distance the print head assembly travels during deceleration must be the same as the distance traveled during acceleration in order to accurately align the character dot columns from one line to the next.

E. Stepper Motor Predetermined Time Constant

Whenever the stepper motor is stepped, or energized, the angular velocity of the rotor is greater than the constant speed which is ultimately required. This is called "overshoot." The frictional load of the carriage assembly (motor rotor, reduction gears, drive belt and print head assembly, or paper feed sprocket shaft and wheels) provides damping or frictional load. Damping slows the motor to less than the required constant speed and is called "undershoot" (see Figure 13, Carriage Stepper Motor Drive Timing). A constant rate of speed is achieved through the averaging of the overshoot and undershoot within each step.

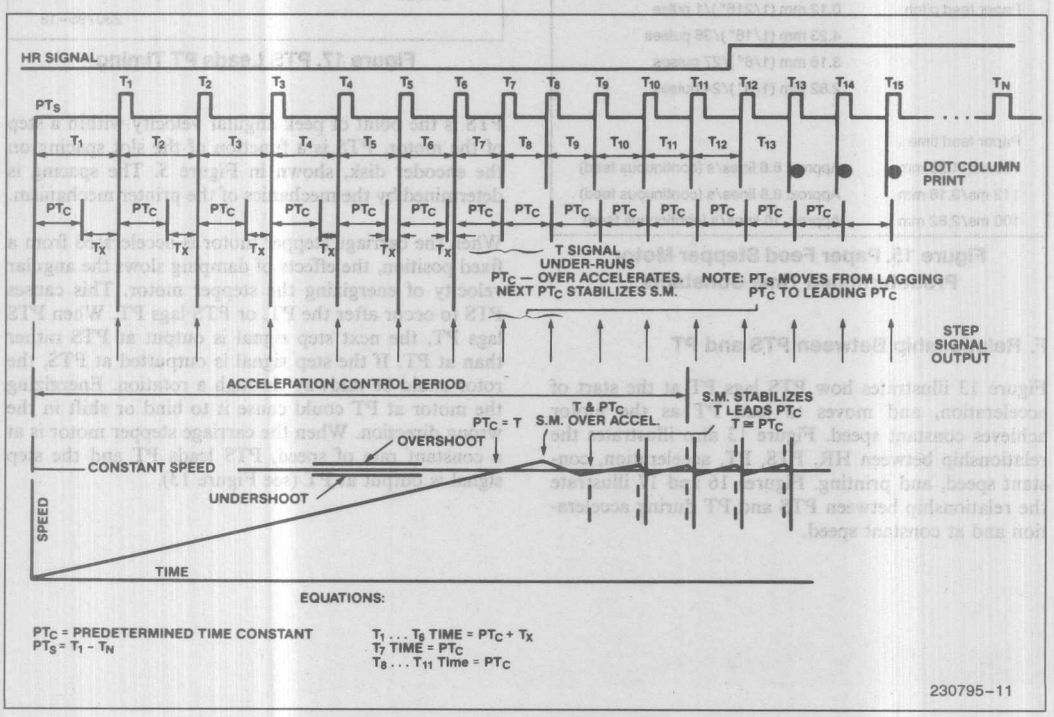


Figure 13. Carriage Stepper Motor Drive Timing

The Predetermined Time (PT) Constant is the time required to average the overshoot and undershoot of the particular stepper motor for a desired constant rate of speed. The PT also is the time required to move the print head assembly a specific distance, accounting for both overshoot and undershoot of the stepper motor.

Changing the Predetermined Time Constant changes the angular displacement of the stepper motor rotor, this in turn changes the output. Figure 14 lists the Time Constants for both standard and condensed character printing. Figure 15 lists the paper feed stepper motor Time Constants used for various line spacing formats. This Application Note implements standard character print and paper feed (6 lines per inch) Time Constants. See Appendix B, Printer Enhancement, for details on implementing non-standard Time Constants.

Character mode	Predetermined time	
Standard or Enlarged Character	2.08 ms	+10% −4%
Condensed Character	4.16 ms	+10% −4%

Figure 14. Carriage Stepper Motor Predetermined Time Constants

Paper feed pitch	0.12 mm (1/216″)/1 pulse
	4.23 mm (1/16″)/36 pulses
	3.18 mm (1/8″)/27 pulses
	2.82 mm (1/9″)/24 pulses
Paper feed time	
150 ms/4.23 mm	Approx. 6.6 lines/s (continuous feed)
113 ms/3.18 mm	Approx. 8.8 lines/s (continuous feed)
100 ms/2.82 mm	Approx. 10 lines/s (continuous feed)

Figure 15. Paper Feed Stepper Motor Predetermined Time Constants

F. Relationship Between PTS and PT

Figure 13 illustrates how PTS lags PT at the start of acceleration, and moves to lead PT as the motor achieves constant speed. Figure 13 also illustrates the relationship between HR, PTS, PT, acceleration, constant speed, and printing. Figures 16 and 17 illustrate the relationship between PTS and PT during acceleration and at constant speed.

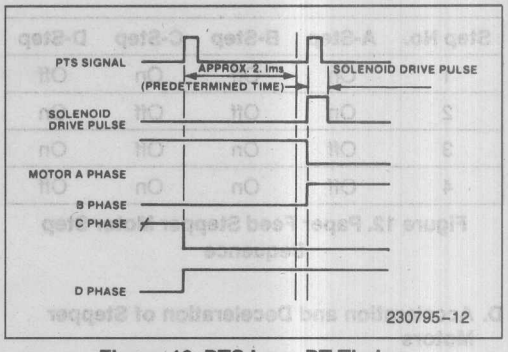


Figure 16. PTS Lags PT Timing

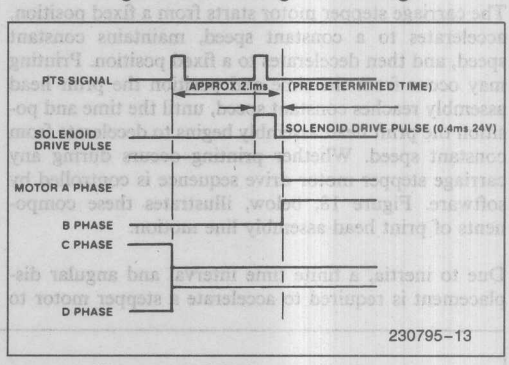


Figure 17. PTS Leads PT Timing

PTS is the point of peek angular velocity within a step of the motor. PTS is a function of the slot spacing on the encoder disk, shown in Figure 5. The spacing is determined by the mechanics of the printer mechanism.

When the carriage stepper motor is accelerated from a fixed position, the effects of damping slows the angular velocity of energizing the stepper motor. This causes PTS to occur after the PT, or PTS lags PT. When PTS lags PT, the next step signal is output at PTS rather than at PT. If the step signal is outputted at PTS, the rotor could be midway through a rotation. Energizing the motor at PT could cause it to bind or shift in the wrong direction. When the carriage stepper motor is at a constant rate of speed, PTS leads PT and the step signal is output at PT (see Figure 13).

G. Stored Time Constants

The time between each step, for a constant number of steps, required for the motor to reach a constant speed, is calculated and stored in Data Memory during acceleration. The values stored are used, in reverse order, during deceleration as the Predetermined Time (PT) Constants. This ensures that the acceleration and deceleration distance traveled by the print head assembly is the same, and that it accurately aligns character dot columns from one line to the next during printing. The time values stored are called "Stored Time Constants." Steps T1 through T11 in Figure 13, represent the Stored Time Constants.

The equations for the Stored Time Constants are given at the bottom of Figure 13, Carriage Stepper Motor Drive Timing.

H. Print Head Assembly "Home" Position

The "logical" Home position for the print head assembly is the left-most position at which printing begins (for L-to-R motion) or ends (for R-to-L motion). The "physical" Home position is the logical HOME position, plus the distance required by the carriage stepper motor to fully accelerate the print head assembly to a constant speed. Printing can only occur when the print head is moving at a constant speed. The printer mechanism manual stipulates eleven step time periods are required to ensure the print head assembly is at a constant speed. These eleven step time periods are the Stored Time Constants described above. Figure 18 illustrates the components of print head assembly line motion and character printing.

SOFTWARE

Introduction

The software description is presented in three sections. First, a brief overview of the software to familiarize the reader with the interdependencies and overall program flow. Second, data and program memory allocation and status register flag definitions. And third, each of the ten software blocks is presented with its own flowchart.

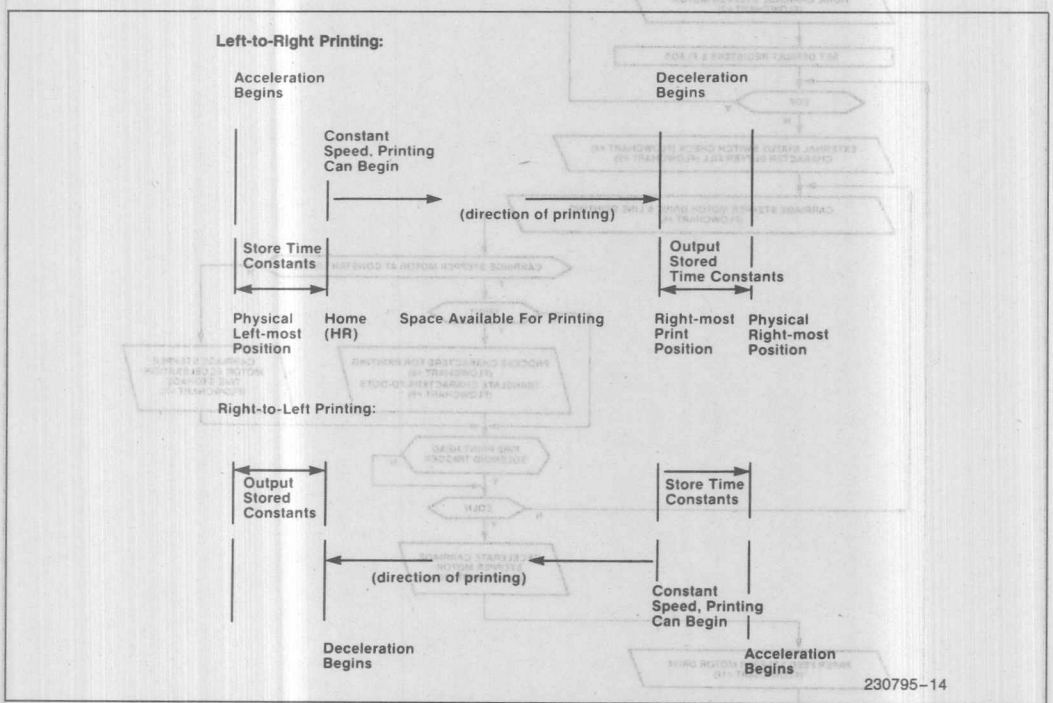


Figure 18. Components of Print Head Assembly Line Motion and Printing

Software Overview

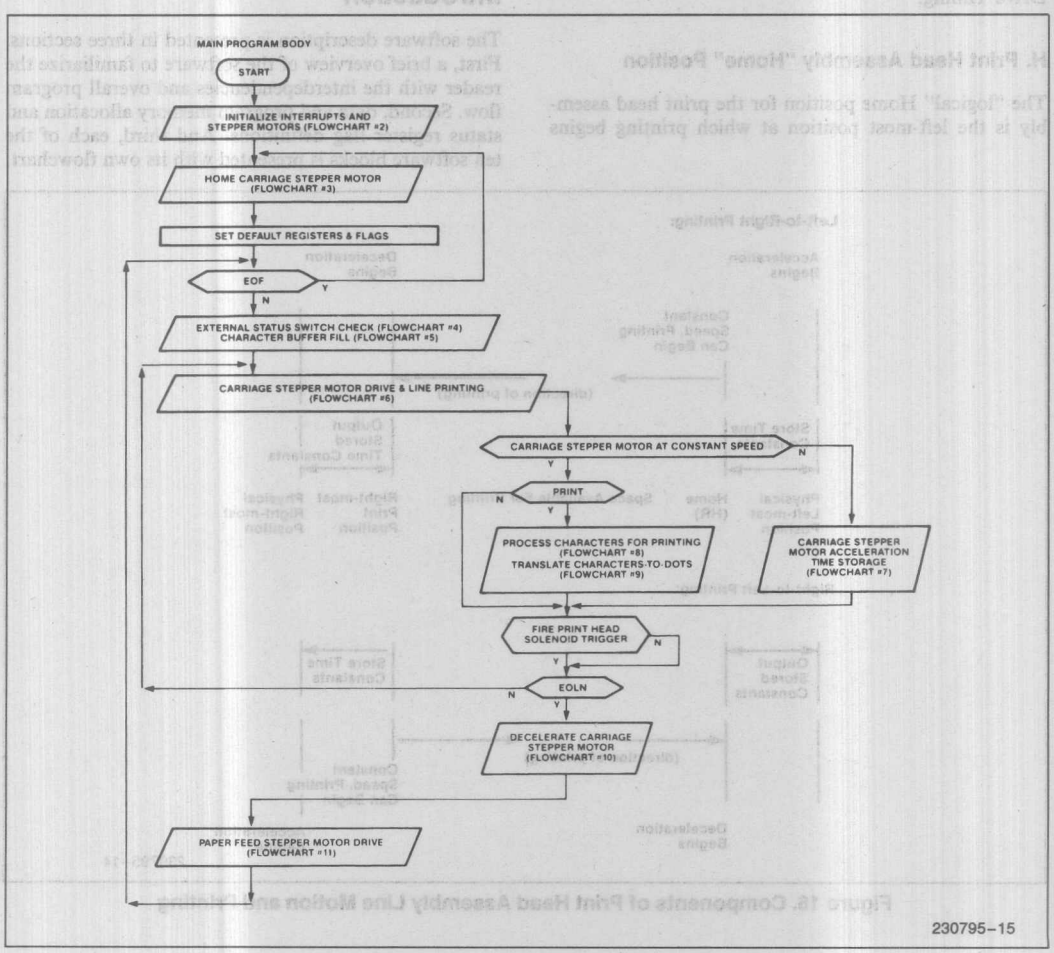
The software is written in Intel UPI-41AH/42AH Assembly Language. A block structure approach is used for ease of development, maintenance, and comprehension. The software is divided into five principal parts.

1. Initialization
2. Character Buffering or Input
3. Stepper Motor Drive and Control
4. Character Processing
5. Character Printing or Output

The five principal parts are incorporated into ten software blocks, listed below.

1. Power On/Reset Initialization
2. Home Print Head Assembly
3. External Status Switch Check
4. Character Buffer Fill
5. Carriage Stepper Motor Drive and Line Printing
6. Accelerate Stepper Motor Time Storage
7. Process Characters for Printing
8. Translate Character-to-Dots
9. Decelerate Carriage Stepper Motor
10. Paperfeed Stepper Motor Drive

Flow Chart No. 1 illustrates the overall software algorithm. Below, is a description of the algorithm.



Flow Chart No. 1. Main Program Body

Upon power-on reset, a software and hardware initialization is performed. This stabilizes and sets inactive the printer hardware and electronics. The print head assembly is then moved to establish its HOME position. The default status registers are set for character buffering, carriage, and paper feed stepper motor drive. The External Status switches are checked; FORMFEED, LINEFEED, ON/OFF LINE, and Character Print TEST. If the printer is ON LINE, the software will loop on filling the Data Memory Character Buffer.

Character or data input to the UPI-42AH is interrupt driven. Characters sent by the host system set the Input Buffer Full (IBF) interrupt and the IBF Program Status flag. Character input servicing (completed during the paper feed and carriage stepper motor drive end Delay subroutine) tests for various ASCII character codes, loads characters into the Character Buffer (CB), and repeats until one of several conditions sets the CB Full status flag. Once the CB Full flag is set, further character transmission by the host system is inhibited and printing can begin.

The carriage stepper motor is initialized, and drive begins for the direction indicated. The motor is accelerated to constant speed, printable character codes are translated to dot patterns and printed (if printing is enabled), and the motor is decelerated to a stop. Two timing loops guarantee both constant speed and protection (Failsafe Time) against stepper motor burn out due to high current overload. The two optical sensors, described in the Printer Mechanism section above, are constantly monitored to maintain constant speed, and trigger print head solenoid firing.

Once the line is printed and the carriage stepper motor drive routine has been completed, a Linefeed is forced. The paper feed stepper motor drive subroutine tests the number of lines to move, and energizes the paper feed stepper motor for the required distance. The lines per page default is 66; if 66 lines have been received, a Formfeed to Top-of-Next-Page is performed. The Top-Of-Page is set at Power On/Reset.

When the EOF code is received, the EOF status flag is set. When the last line has been printed, the EOF check will force the print head assembly to the HOME position. The EOF flag is tested following each paperfeed stepper motor drive. The next entry to the External Status Check subroutine begins a loop which waits for input from either the external status switches or the host system.

The software character dot matrix used in this application is 5 x 7 of the available 9 x 9 print head solenoid matrix. Although lower case descenders and block/line graphics characters are not implemented, Appendix B, Printer Enhancements, discusses how and where these enhancements could be added. The software implements the full 95 ASCII printable characters set.

Memory and Register Allocation

DATA MEMORY ALLOCATION (RAM)

The UPI-42AH has 256 bytes of Data Memory. Sixteen bytes are used by the two 8 byte register banks (RB0 and RB1). Sixteen additional bytes are used for the Program Stack. The Stored Time Constants utilize 11 bytes, while the stepper motor phase storage requires 4 bytes. Below is a detailed description of Data and Program Memory.

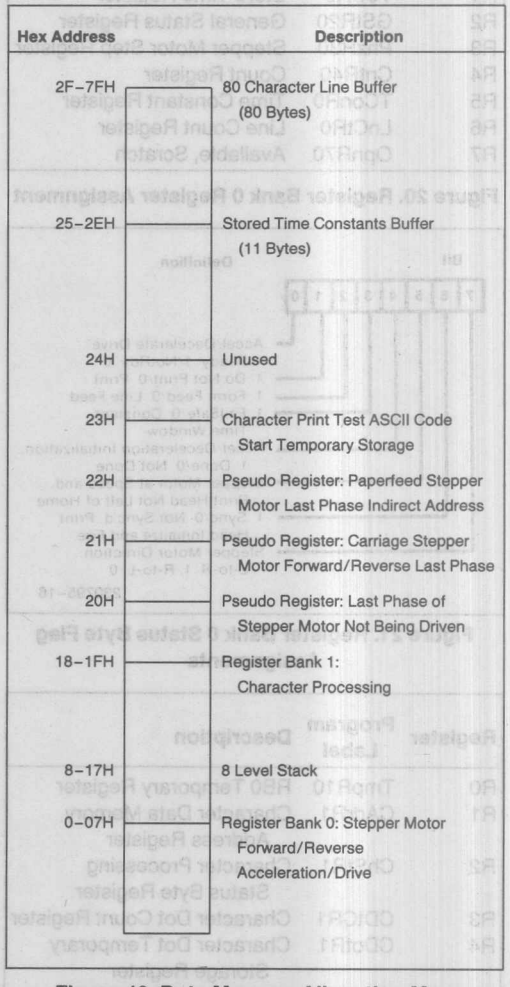


Figure 19. Data Memory Allocation Map

Register Bank 0 is used for stepper motor drive functions. Register Bank 1 is used for character processing. Each register bank's register assignments is listed in Figures 20 and 22, respectively. Each register bank has one register allocated as a Status Register. Figures 21 and 23 detail the Status Register flag assignments. Note

that bit 7 of each Status Byte is used as a print head assembly motion direction flag. This saves coding of the Select Register Bank (SEL RBn) instruction at each point the flag is checked.

Register Bank 0		
Register	**Program Label**	**Description**
R0	TmpR00	RB0 Temporary Register
R1	TStrR0	Store Time Register
R2	GStR20	General Status Register
R3	PhzR30	Stepper Motor Step Register
R4	CntR40	Count Register
R5	TConR0	Time Constant Register
R6	LnCtR0	Line Count Register
R7	OpnR70	Available, Scratch

Figure 20. Register Bank 0 Register Assignment

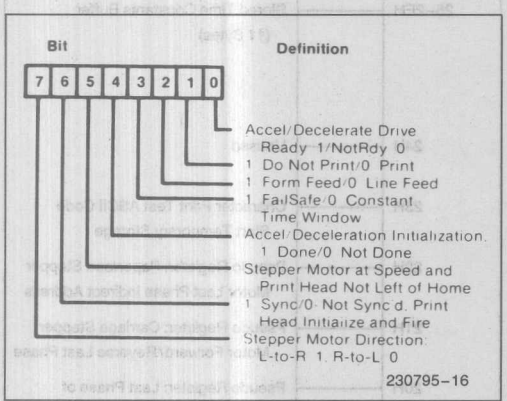


Figure 21. Register Bank 0 Status Byte Flag Assignments

Register	Program Label	Description
R0	TmpR10	RB0 Temporary Register
R1	CAdrR1	Character Data Memory Address Register
R2	ChStR1	Character Processing Status Byte Register
R3	CDtCR1	Character Dot Count Register
R4	CDotR1	Character Dot Temporary Storage Register
R5	CCntR1	Character Count Temporary Register
R6	StrCR1	Store Character Register
R7	OpnR71	Available/Scratch

Figure 22. Register Bank 1 Register Assignment

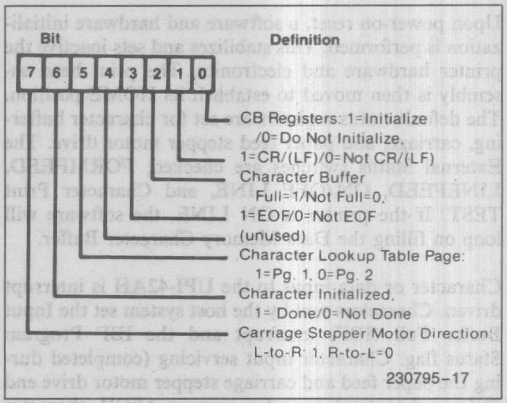


Figure 23. Register Bank 1 Status Byte Flag Assignments

PROGRAM MEMORY ALLOCATION (EPROM/ROM)

The UPI-42AH has 2048 bytes of Program Memory divided into eight pages, each 256 bytes. Figure 24 illustrates the Program Memory allocation map by page.

Page	Hex Address	Description
Page 7	1792–2047	Character to Dot Pattern Lookup Table; Page 2: ASCII 50H–7EH
Page 6	1536–1791	Character to Dot Pattern Lookup Table; Page 1: ASCII 20H–4FH (sp-M)
Page 5	1280–1535	Miscellaneous Subroutines: InitAl/AllOff Clear Data Memory Home Print Head Assembly Character Print Test Initialize Carriage Stepper Motor Delay Stepper Motor Deselect
Page 4	1024–1279	Paper Feed Stepper Motor Drive
Page 3	768–1023	Stepper Motor Step Look Up Table (Indexed) Character Processing and Translation Print Head Firing
Page 2	51–767	Carriage Stepper Motor Acceleration Time Calculation and Storage Stepper Motor Deceleration
Page 1	256–511	Carriage Stepper Motor Drive
Page 0	0–255	Initialization–Jump-on-Reset Main Program Body External Status Switch Check Character Buffer Fill

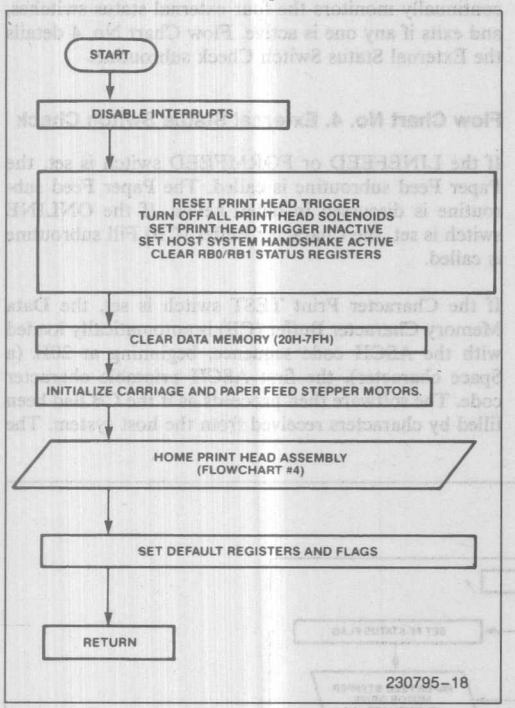
Figure 24. Program Memory Allocation Map

Software Functional Blocks

Below is a description and flow chart for each of the ten software blocks listed.

1. Power-On/Reset Initialization

The first operational part in Flow Chart No. 1 is the Power-On or Reset Initialization. Flowchart No. 2 illustrates the Initialization sequence in detail.



Flow Chart No. 2. Power-On/Reset Initialization

Initialization first disables both interrupts. This is done as a precaution to prevent the system software from hanging-up should an interrupt occur before the proper registers and Data Memory values are initialized.

Initialization then deactivates the system electronics. This is also a precaution to protect the printer mechanism and includes the print head solenoid (trigger and data) lines and the stepper motor select lines. The host system handshake signals are activated to inhibit data transfer from the host until the printer is ready to accept data.

Next, Data Memory is cleared from 20H to 7FH. This includes the 80 byte Character Buffer, the 11 byte Stored Time Constants buffer, and the 4 bytes used as pseudo registers. The pseudo registers are Data Memory locations used as if they were registers. They serve as storage locations for step data used in accurately reversing the direction of the carriage stepper motor, and stabilizing either of the stepper motors not being driven.

The Data Memory locations 00H through 1FH are not cleared. These locations are Register Bank 0 (00H–07H), Program Stack (08H–17H), and Register Bank 1 (18H–1FH) (see Figure 19). Clearing the Program Registers or Stack would cause the initialization subroutine to become lost. The registers are used from the beginning of the program. Care is taken to initialize the registers and stack accurately prior to each program subroutine as required.

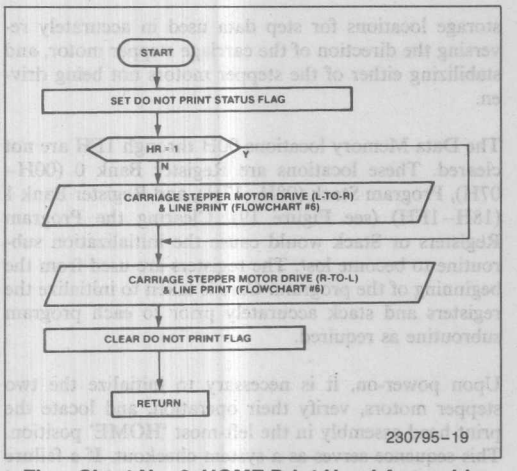
Upon power-on, it is necessary to initialize the two stepper motors, verify their operation, and locate the print head assembly in the left-most 'HOME' position. This sequence serves as a system checkout. If a failure occurs, the motors are deselected and the external status light is turned on. Each stepper motor is selected and energized for a sequence of four steps. This serves to align and stabilize each stepper motor's rotor position, preventing the rotor from skipping or binding when the first drive sequence begins.

At the end of each stepper motor's initialization, the last step data address is stored in one of the Data Memory pseudo registers. The last step data address is recalled at the beginning of the next corresponding stepper motor drive sequence, and used as the basis of the next step sequence. This ensures that the stepper motor always receives the exact next step data, in sequence, to guarantee smooth stepper motor motion. This also guarantees the motor never skips or jerks, which would misalign the start, stop, and character dot column positions. A stepper motor not being driven has its last phase data output held constant to stabilize it.

Following any stepper motor drive sequence of either motor, a delay of 30–60 ms occurs by switching the current to Hold Current, stabilizing the motor before it is deselected.

2. Home Print Head Assembly

At the end of the carriage stepper motor four step initialization, the output of the HR optical sensor is tested. The level of the HR signal indicates which drive sequence will be required to 'HOME' the print head assembly. If the print head assembly is to the right of HR, HR is high, the print head assembly need only be moved to from Right-to-Left until HR is low, then decelerated to locate the physical home position. If HR is low, the print head assembly must be moved first Left-to-Right until HR is high, then Right-to-Left to locate both the logical and physical 'HOME' positions. In each case, the software accelerates the carriage stepper motor, generating the Stored Time Constants then decelerates the stepper motor using the Stored Time Con-



Flow Chart No. 3. HOME Print Head Assembly

stants (see Background section above). Flow Chart No. 3 details the HOME print head assembly subroutine. Figures 13 and 18 illustrate the components of acceleration and print head assembly line motion.

The carriage stepper motor drive subroutines used to HOME the print head assembly and to print, are the same. A status flag, called Do-Not-Print, determines whether the Character Processing subroutine is called. The flag is set by the subroutine which calls the Carriage Stepper Motor Drive subroutine. Details of the carriage and paper feed stepper motor drive and character processing subroutines are covered separately below.

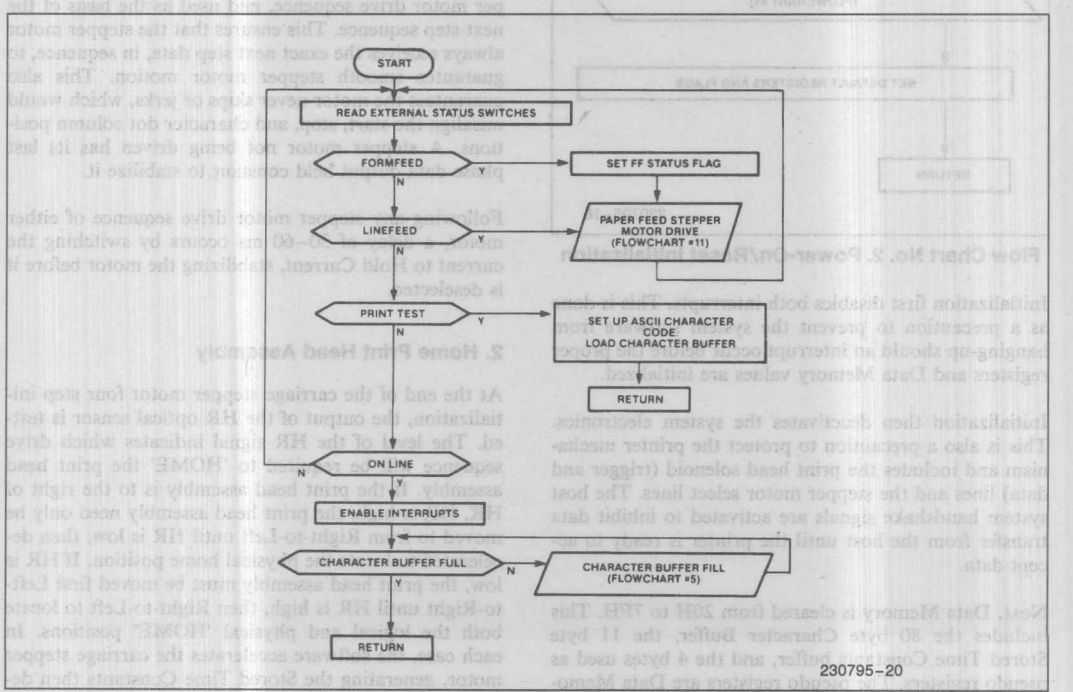
3. External Status Switch Check

Once the system is initialized and the print head is at the HOME position, the software enters a loop which continually monitors the four external status switches, and exits if any one is active. Flow Chart No. 4 details the External Status Switch Check subroutine.

Flow Chart No. 4. External Status Switch Check

If the LINEFEED or FORMFEED switch is set, the Paper Feed subroutine is called. The Paper Feed subroutine is discussed in detail below. If the ONLINE switch is set, the Character Buffer (CB) Fill subroutine is called.

If the Character Print TEST switch is set, the Data Memory Character Buffer (CB) is automatically loaded with the ASCII code sequence, beginning at 20H (a Space character), the first ASCII printable character code. The software then proceeds as if the CB had been filled by characters received from the host system. The



Flow Chart No. 4. External Status Switch Check

External Status Switch Check subroutine is exited and character printing begins. When the line has finished printing, a linefeed occurs (as shown in the main program Flow Chart No. 1) and the program returns to the External Status Switch Check subroutine. If the TEST switch remains active, the ASCII character code is incremented and program continues as before. This will eventually print all 95 ASCII printable characters. An example of the TEST printer output, the complete ASCII character code printed, is shown in Figure 25.

4. Character Buffer Fill

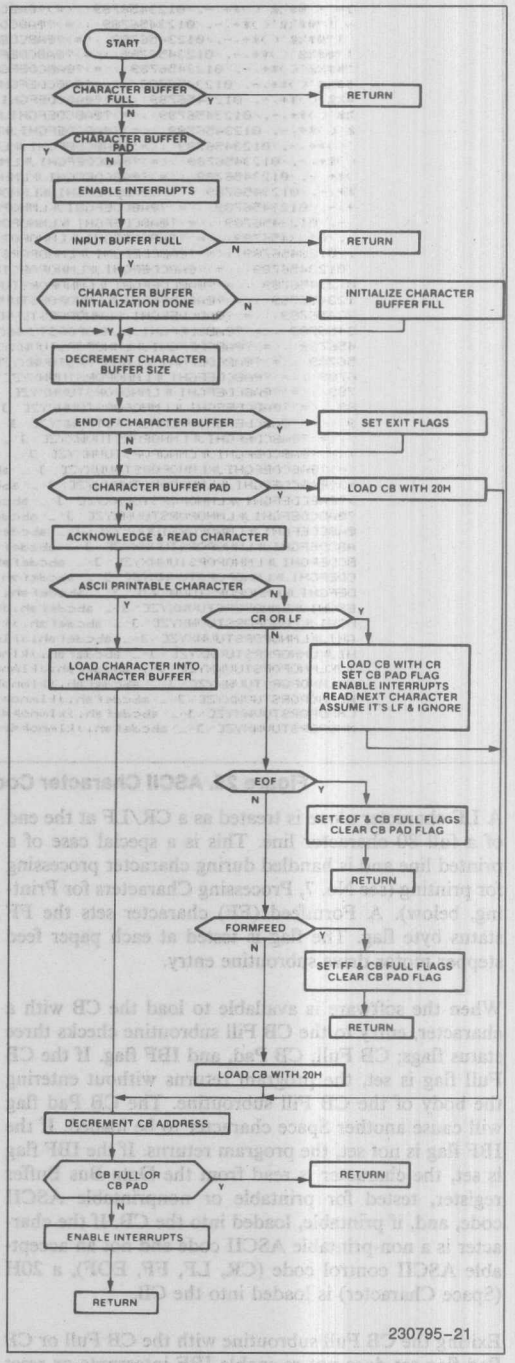
The Character Buffer (CB) Fill subroutine is called from three points within the main program; External Status Switch subroutine, and the Delay subroutine following the carriage and paper feed stepper motor drive subroutines. Flowchart No. 5 details the Character Buffer Fill subroutine operation.

The approximate 80 ms total pre-deselect delay at the end of each stepper motor drive sequence, 40 ms carriage and 40 ms paper feed stepper motor pre-deselect delay, is sufficient to load an entire 80 character line. Half the CB is filled at the end of printing the current line, and the second half is filled at the end of a paper feed. There is no time lost in printing throughput due to filling the character buffer.

Character input is interrupt driven. When the IBF interrupt is enabled, a transmitted character sets the IBF interrupt and IBF Program Status flag. Three instructions make up the IBF interrupt service routine. This short routine disables further interrupts, sets the BUSY handshake line active, inhibiting further transmission by the host, and returns. The subroutine can be executed at virtually any point in the software flow without affecting the printer mechanism operation. Processing of the received character takes place during one of the three program segments mentioned above. The BUSY line remains active until the character is processed by the CB Fill subroutine.

The CB is 80 bytes from the top of Data Memory (30H–7FH). It is a FIFO for forward, left-to-right printing, and a LIFO for reverse, right-to-left, printing. Loading the CB always begins at the top, 7FH. One character may be loaded into the buffer each time the CB Fill subroutine is called.

The CB is always filled with 80 bytes of data prior to printing. If the total number of characters input up to a Carriage Return (CR)/Linefeed (LF), does not completely fill the CB, the CR code is loaded into the CB and the balance of the CB is padded with 20H (Space Character) until the CB is full. A Linefeed (LF) character following a Carriage Return is ignored. A LF is always forced at the end of a printed line. When the CB is full, the CB Full status byte flag is set and printing can begin.



Flow Chart No. 5. Character Buffer Fill

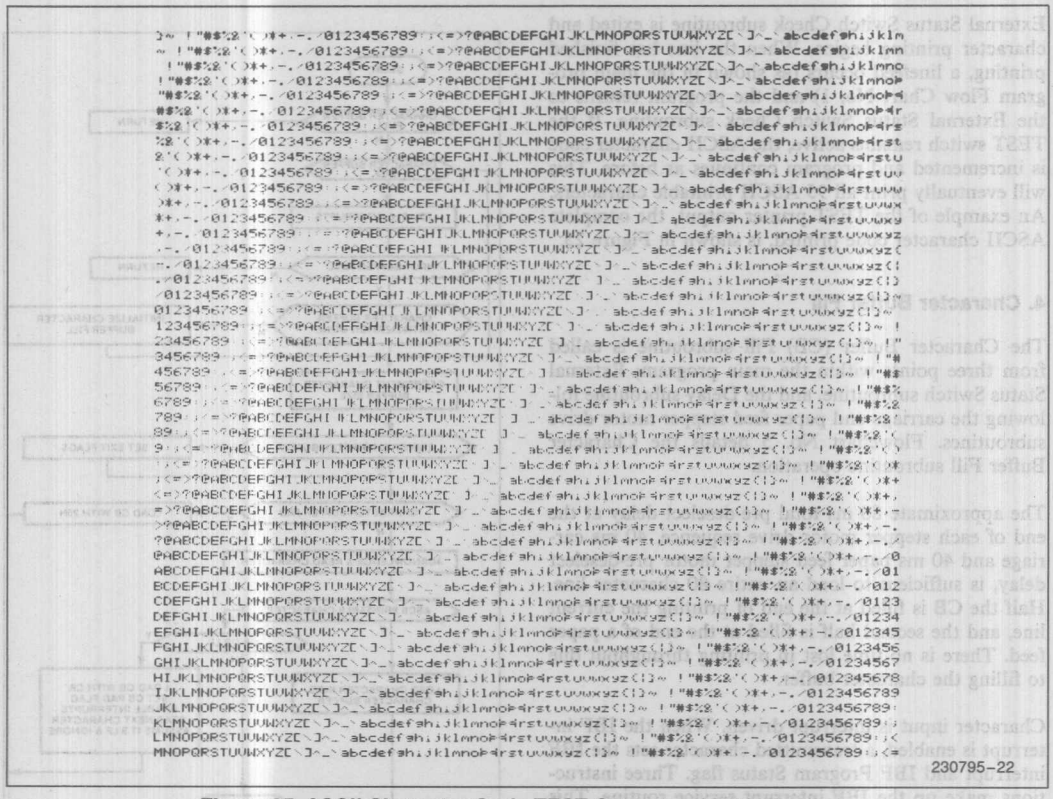


Figure 25. ASCII Character Code TEST Output and Print Example

A LF character alone is treated as a CR/LF at the end of a full 80 character line. This is a special case of a printed line and is handled during character processing for printing (see No. 7, Processing Characters for Printing, below). A Formfeed (FF) character sets the FF status byte flag. The flag is tested at each paper feed stepper motor drive subroutine entry.

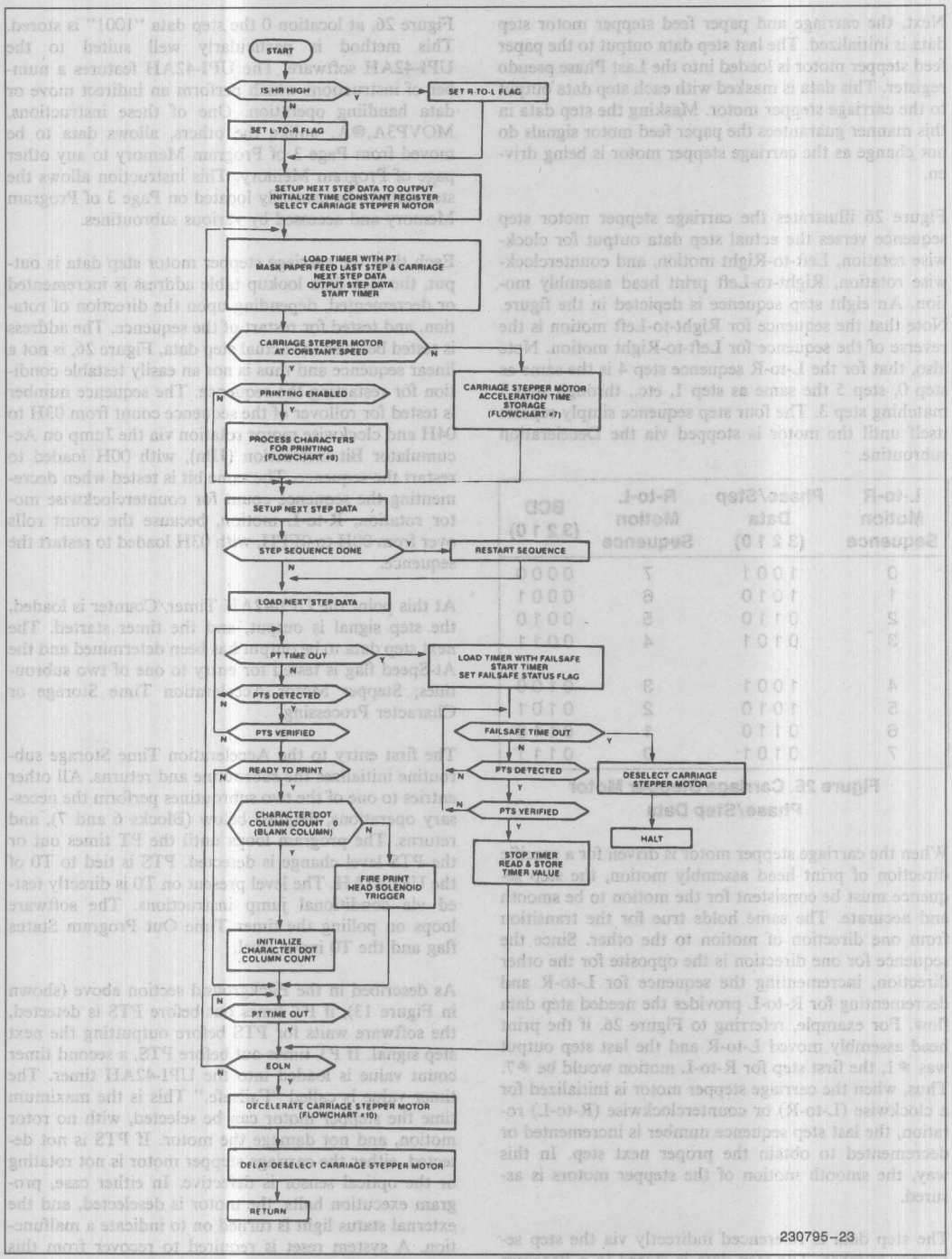
When the software is available to load the CB with a character, entry to the CB Fill subroutine checks three status flags; CB Full, CB Pad, and IBF flag. If the CB Full flag is set, the program returns without entering the body of the CB Fill subroutine. The CB Pad flag will cause another Space character to be loaded. If the IBF flag is not set, the program returns. If the IBF flag is set, the character is read from the Data Bus Buffer register, tested for printable or nonprintable ASCII code, and, if printable, loaded into the CB. If the character is a non-printable ASCII code and not an acceptable ASCII control code (CR, LF, FF, EOF), a 20H (Space Character) is loaded into the CB.

Exiting the CB Full subroutine with the CB Full or CB Pad flag set does not re-enable IBF interrupts or reset the BUSY line. If neither of these flags is set, exiting the CB Fill subroutine sets BUSY inactive and IBF interrupts are enabled. Once the CB Full status byte flag is set, IBF interrupts are disabled until the CB has been entirely emptied, the line printed, or the system Reset.

5. Carriage Stepper Motor Drive and Line Printing

The carriage stepper motor drive subroutine controls both L-to-R and R-to-L print head assembly motion. Upon entering the subroutine, the HR signal level is tested to determine the direction of print head assembly motion and the Direction status flag is set. The default control register values are loaded and balance of the default status flags are set for stepper motor control and character processing. The default control register values include PT and the step sequence look-up table start address for the direction indicated.

The direction flag is tested throughout the carriage stepper motor drive and character processing subroutines. This enables the same subroutines to control activities for either direction, simplifying and shorting the overall program. Flow Chart No. 6 illustrates the carriage stepper motor drive subroutine.



Flow Chart No. 6. Carriage Stepper Motor Drive/Line Printing

Next, the carriage and paper feed stepper motor step data is initialized. The last step data output to the paper feed stepper motor is loaded into the Last Phase pseudo register. This data is masked with each step data output to the carriage stepper motor. Masking the step data in this manner guarantees the paper feed motor signals do not change as the carriage stepper motor is being driven.

Figure 26 illustrates the carriage stepper motor step sequence verses the actual step data output for clockwise rotation, Left-to-Right motion, and counterclockwise rotation, Right-to-Left print head assembly motion. An eight step sequence is depicted in the figure. Note that the sequence for Right-to-Left motion is the reverse of the sequence for Left-to-Right motion. Note also, that for the L-to-R sequence step 4 is the same as step 0, step 5 the same as step 1, etc., through step 7 matching step 3. The four step sequence simply repeats itself until the motor is stopped via the Deceleration subroutine.

L-to-R Motion Sequence	Phase/Step Data (3 2 1 0)	R-to-L Motion Sequence	BCD (3 2 1 0)
0	1 0 0 1	7	0 0 0 0
1	1 0 1 0	6	0 0 0 1
2	0 1 1 0	5	0 0 1 0
3	0 1 0 1	4	0 0 1 1
4	1 0 0 1	3	0 1 0 0
5	1 0 1 0	2	0 1 0 1
6	0 1 1 0	1	0 1 1 0
7	0 1 0 1	0	0 1 1 1

Figure 26. Carriage Stepper Motor Phase/Step Data

When the carriage stepper motor is driven for a specific direction of print head assembly motion, the step sequence must be consistent for the motion to be smooth and accurate. The same holds true for the transition from one direction of motion to the other. Since the sequence for one direction is the opposite for the other direction, incrementing the sequence for L-to-R and decrementing for R-to-L provides the needed step data flow. For example, referring to Figure 26, if the print head assembly moved L-to-R and the last step output was #1, the first step for R-to-L motion would be #7. Thus, when the carriage stepper motor is initialized for a clockwise (L-to-R) or counterclockwise (R-to-L) rotation, the last step sequence number is incremented or decremented to obtain the proper next step. In this way, the smooth motion of the stepper motors is assured.

The step data is referenced indirectly via the step sequence number. The step data is stored in a Program Memory look-up table whose addresses correspond to the step sequence numbers. For example, as shown in Figure 26, at location 0 the step data "1001" is stored. This method is particularly well suited to the UPI-42AH software. The UPI-42AH features a number of instructions which perform an indirect move or data handling operation. One of these instructions, MOVP3A,@A, unlike the others, allows data to be moved from Page 3 of Program Memory to any other page of Program Memory. This instruction allows the step data to be centrally located on Page 3 of Program Memory and accessed by various subroutines.

Each time the carriage stepper motor step data is output, the step data lookup table address is incremented or decremented, depending upon the direction of rotation, and tested for restart of the sequence. The address is tested because the actual step data, Figure 26, is not a linear sequence and thus is not an easily testable condition for restarting the sequence. The sequence number is tested for rollover of the sequence count from 03H to 04H and clockwise motor rotation via the Jump on Accumulator Bit instruction (JBn), with 00H loaded to restart the sequence. The same bit is tested when decrementing the sequence count for counterclockwise motor rotation, R-to-L motion, because the count rolls over from 00H to 0FFH, with 03H loaded to restart the sequence.

At this point the UPI-42AH Timer/Counter is loaded, the step signal is output, and the timer started. The next step data to be output has been determined and the At-Speed flag is tested for entry to one of two subroutines; Stepper Motor Acceleration Time Storage or Character Processing.

The first entry to the Acceleration Time Storage subroutine initializes the subroutine and returns. All other entries to one of the two subroutines perform the necessary operations, detailed below (Blocks 6 and 7), and returns. The program loops until the PT times out or the PTS level change is detected. PTS is tied to T0 of the UPI-42AH. The level present on T0 is directly tested via conditional jump instructions. The software loops on polling the timer Time Out Program Status flag and the T0 input level.

As described in the Background section above (shown in Figure 13), if PT times out before PTS is detected, the software waits for PTS before outputting the next step signal. If PT times out before PTS, a second timer count value is loaded into the UPI-42AH timer. The timer value is called "Failsafe." This is the maximum time the stepper motor can be selected, with no rotor motion, and not damage the motor. If PTS is not detected, either the carriage stepper motor is not rotating or the optical sensor is defective. In either case, program execution halts, the motor is deselected, and the external status light is turned on to indicate a malfunction. A system reset is required to recover from this condition. The Failsafe time is approximately 20 milliseconds, including PT.

The Failsafe time loop also serves as a means of tracking the elapsed time between PT time out and PTS. Entry to the Failsafe time loop sets the Failsafe/Constant Time Window status flag. This flag is tested by the Acceleration Time Storage subroutine for branching to the proper time storage calculation to be performed (see Figure 13 and Block 6 below for further description).

During the Failsafe timer loop, if PTS is detected and verified as true, the Failsafe timer value is read and stored in the Time Storage register. This value is used during the next Acceleration Time Storage subroutine call to calculate the Stored Time Constant (see Block 6 below). If PTS is invalid, the flow returns to the timer loop just exited, again waiting for PTS or Failsafe time out.

During the PT time loop, if PTS is detected and verified, the Sync flag is tested for entry to the print head solenoid firing subroutine. This flag is set by the first entry to the Character Processing subroutine. The flag synchronizes the solenoid firing with character processing. Only if characters are processed for printing will the solenoids be enabled, via the Sync flag, for firing. This prevents the solenoids from being fired without valid character dot data present.

As described in the Background section "Relationship Between PTS and PT," PTS is the point of peek angular velocity within a step of the motor. After PTS is detected the motor speed ramps down, compensating for the overshoot of the rotor motion. PTS is the optimum time for print head solenoid firing, as shown in Figure 13. This is the stable point of motor rotation and, thus, the print head assembly motion. If PTS is detected during PT, printing is enabled, the Sync flag is set, and the solenoid trigger is fired.

The firing of the solenoid trigger, following PTS, is very time critical. The time between PTS and solenoid firing must be consistent for accurate dot column alignment throughout the printed line. The software is designed to meet this requirement by placing all character processing and motor control overhead before the solenoid firing subroutine is called. The actual instruction sequence which fires the print head solenoid trigger is plus or minus one instruction for any call to the subroutine.

Once the timer loop is complete, the software tests for Exit conditions. If the Exit conditions fail, the software loops to output the next step signal, starts the PT timer, and continues to accelerate the carriage stepper motor, or process, and print characters. If the Exit test is true, the carriage stepper motor is decelerated to a fixed position, and the program returns to the main program flow (see Flowchart 1).

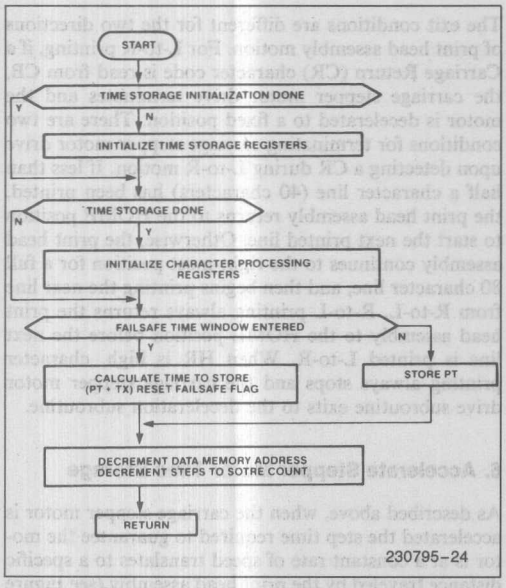
The exit conditions are different for the two directions of print head assembly motion. For L-to-R printing, if a Carriage Return (CR) character code is read from CB, the carriage stepper motor drive terminates and the motor is decelerated to a fixed position. There are two conditions for terminating carriage stepper motor drive upon detecting a CR during L-to-R motion. If less than half a character line (40 characters) has been printed, the print head assembly returns to the HOME position to start the next printed line. Otherwise, the print head assembly continues to the right-most position for a full 80 character line, and then begins printing the next line from R-to-L. R-to-L printing always returns the print head assembly to the HOME position before the next line is printed L-to-R. When HR is high, character printing always stops and the carriage stepper motor drive subroutine exits to the deceleration subroutine.

6. Accelerate Stepper Motor Time Storage

As described above, when the carriage stepper motor is accelerated the step time required to guarantee the motor is at a constant rate of speed translates to a specific distance traveled by the print head assembly (see Figure 18). In order to position the print head assembly accurately for bi-directional printing, the distance traveled during deceleration must be the same as during acceleration. The Carriage Motor Acceleration Time Storage subroutine calculates the step times needed to accelerate the carriage stepper motor, and stores them in Data Memory for use as PT during deceleration.

The first call of the Carriage Stepper Motor Acceleration Time Storage subroutine initializes the required registers and status flags. The time calculation begins with the second carriage stepper motor step signal output. The program returns to the carriage stepper motor drive subroutine and loops on PT. Each subsequent call of the Acceleration Time Storage subroutine tests the Failsafe/Constant flag and branches accordingly (see Flow Chart 7). The Acceleration Time Storage subroutine has two parts which correspond to PTS leading or PTS lagging PT.

If the Failsafe/Constant flag is set, PTS lagged PT. The time from PT time out to PTS, Tx (see Figure 13), must be added to the PT and stored in Data Memory. As described above, if PT lagged PT, the Failsafe time is loaded and PTS is again polled during the time loop. When PTS occurs within the Failsafe time, the timer is stopped and the timer value stored. The UPI-42AH timer is an up timer, which means that the value stored is the time remaining of the Failsafe time when PTS occurred. The elapsed time must be calculated by subtracting the time remaining (the value stored) from the Failsafe time constant. This is done in software by using two's complement arithmetic. If the Failsafe flag is not set PTS led PT, and PT is the Stored Time Constant stored.



Flow Chart No. 7. Carriage Stepper Motor Acceleration Time Storage

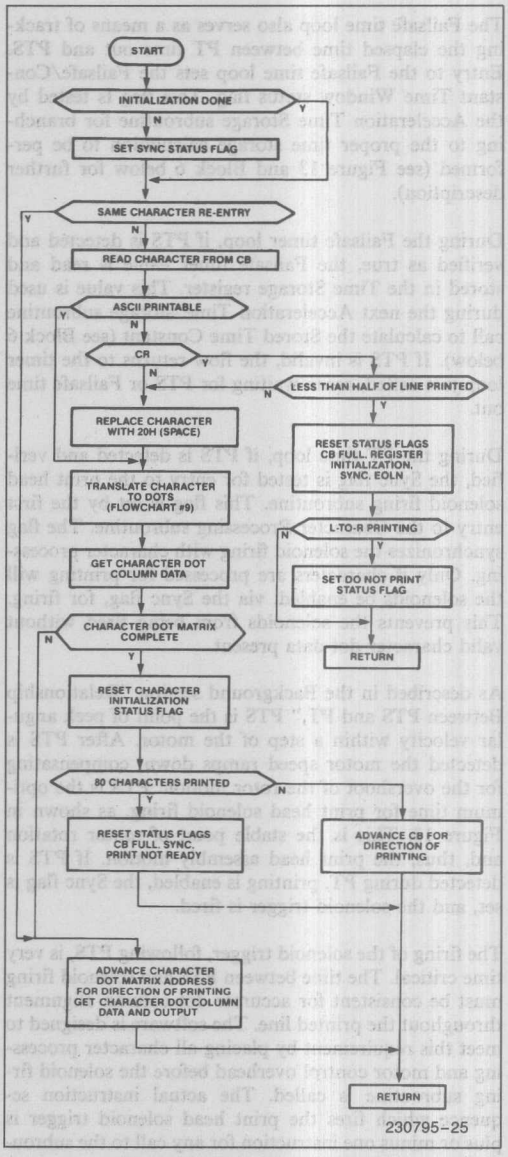
Indirect addressing of Data Memory is used to reference the Stored Time Constant Data Memory location. The Data Memory location address is decremented each time the Acceleration Time Storage subroutine is exited and a Stored Time Constant has been generated.

The last Acceleration Time Storage subroutine exit sets the At-Speed status flag and initializes the character processing registers and flags.

7. Process Characters for Printing

The Character Processing subroutine is entered only if the Home Reset (HR) optical sensor signal is high and printing is enabled. Otherwise, the software simply returns to the Carriage Stepper Motor Drive subroutine. There are two cases when printing is not enabled; during the HOME subroutine operation, and when the print head assembly returns to the HOME position after printing less than half an 80 character line. If printing is enabled, the Sync status flag is set.

All character processing operations use the second UPI-42AH Data Memory Register Bank, RB1. Register Bank 1 is independent of Data Memory Register Bank 0, used for stepper motor control. The use of two independent register banks greatly simplifies the software flow, and helps to ensure the accuracy of event sequences that must be handled in parallel. Each register bank must be initialized only once for any entry to either the Carriage Stepper Motor Drive or Character Processing subroutines. A single UPI-42AH Assembly Language instruction selects the appropriate register



Flow Chart No. 8. Process Characters for Printing

bank. Initializing the character processing registers includes loading the maximum character count (80), dot matrix size count (6), and CB start address. The CB start address is print direction dependent, as described in Block 4, above.

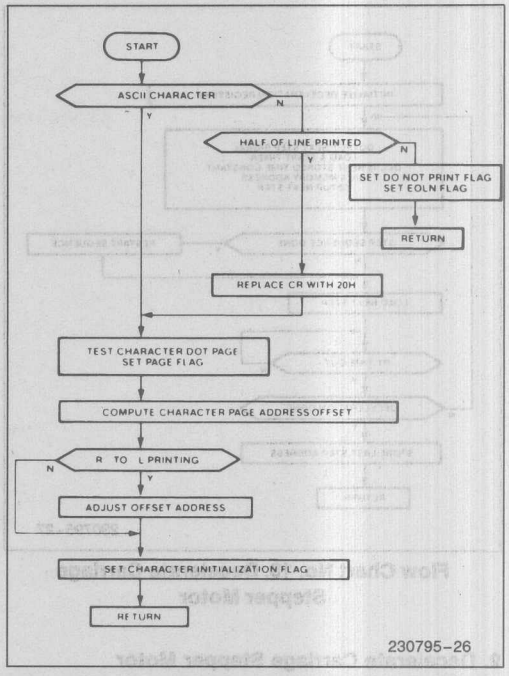
Character processing reads a character from the CB, tests for control codes, translates the character to dots,

and conditionally exits, returning to the Carriage Stepper Motor Drive subroutine. Flow Chart 8 details the character processing subroutine.

Each character requires six steps of the carriage stepper motor to print; five for the 5 character dot columns and 1 for the blank dot column between each character. Reading a character from the CB and character-to-dot pattern translation takes place during the last character dot column, or blank column, time.

The first character line entry to the Character Processing subroutine appears to the software as if a last character dot column (blank column) had been entered. The next character, in this case the first character in the line, is translated and printing can begin. This method of initializing the Character Processing subroutine utilizes the same software for both start-up and normal character flow. Once a character code has been translated to a dot matrix pattern starting address in the look-up table, all subsequent entries to the Character Processing subroutine simply advance the dot column data address and outputs the data.

The decision to translate the character to dots during the blank column time was an arbitary one. As was the choice of the blank column following rather than preceding the actual character dot matrix printing.



Flow Chart No. 9. Translate Character-to-Dots

8. Translate Character-to-Dots

Characters-to-dot pattern translation involves converting the ASCII code into a look-up table address, where the first of the five bytes of character dot column data is stored. The address is then incremented for the next column of dot pattern data until the full character has been printed.

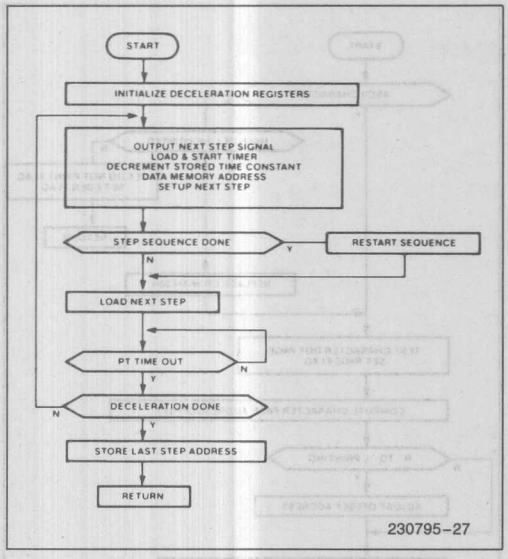
The dot pattern look-up table occupies two pages, or approximately 512 bytes of Program Memory. A printable ASCII character is tested for its dot pattern location page and the offset address, from zero, on that page. Both the page test and page offset calculations use two's complement arithmetic, with a jump on carry or not carry causing the appropriate branching. Once the pattern page and address are determined the indirect addressing and data move instructions are used to read and output the data to the print head solenoids. Flowchart 9 details the Character-to-Dots Translation subroutine.

In the case of R-to-L printing, although the translation operation is the same, the character is printed in reverse. This requires that the character dot pattern address be incremented by five, before printing begins, so that the first dot column data output is the last dot column data of the character. The dot pattern look-up table address is then decremented rather than incremented, as in L-to-R printing, for the balance of the character. Translation still takes place during the last character dot column, the blank column, and the blank column follows the character matrix.

Only one control code, a Carriage Return (CR), is encountered by the character translation subroutine. Linefeed (LF) characters are stripped off by the CB Fill subroutine. If a CR code is detected the software tests for a mid-line exit condition; less than half the line printed exits the stepper motor drive subroutine and HOMEs the print head assembly before printing the next line. If the test fails, more than half the line has been printed, the CR is replaced by a 20H (Space character) and printing continues for the balance of the line; the space characters padding the CB are printed.

As mentioned above, the character dots are printed and the print head trigger is fired when the PTS signal is detected and verified and the carriage stepper motor is At Speed.

When the character to print test fails the CB Buffer size count equals zero, the Carriage Stepper Motor Drive subroutine exit flags are set, and the flow passes to the Deceleration and Delay subroutines and programs returns to the main program flow.



Flow Chart No. 10. Decelerate Carriage Stepper Motor

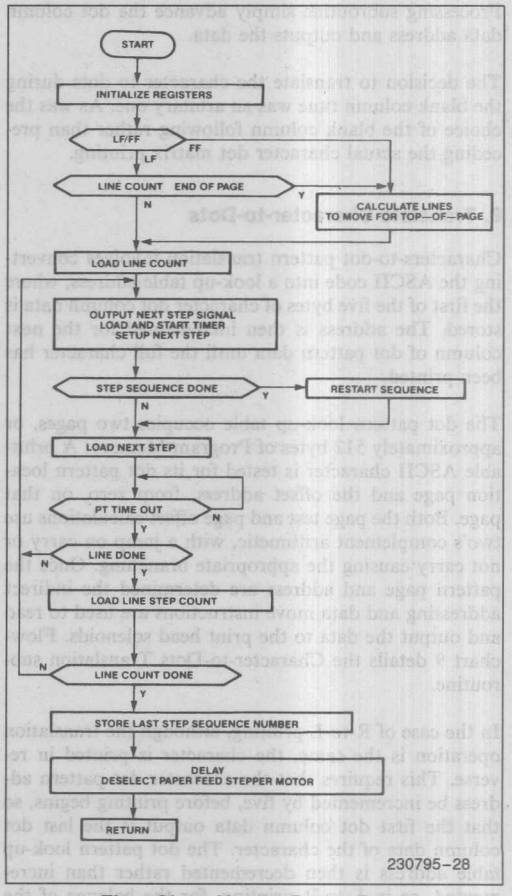
9. Decelerate Carriage Stepper Motor

The transition from the Carriage Stepper Motor Drive subroutine to the Deceleration subroutine outputs the next step signal in sequence, and then initializes the Deceleration subroutine registers; Stored Time Constants Data Memory buffer end address and size. The Stored Time Constant Buffer is a LIFO for deceleration of the carriage stepper motor. The buffer size is used as the step count. When the step count decrements to zero, the step signal output is terminated, and the last step sequence number is stored in the carriage stepper motor Next Step pseudo register. The last step sequence number is recalled, during initialization of the next carriage stepper motor drive, as the basis of the next step data signal to be output. See Flow Chart 10.

When the carriage stepper motor is decelerated, Failsafe protection and PTS monitoring are not necessary. The Deceleration subroutine acts as its own failsafe mechanism. Should the stepper motor hang-up, the subroutine would exit and deselect the motor in sufficient time to protect the motor from burnout. Since neither Failsafe nor print head solenoid firing take place during deceleration, PTS is not needed. PT is replaced by the Stored Time Constant values in Data Memory. The Deceleration subroutine determines the next step signal to output, loads the Timer with the Stored Time Constant, starts the UPI-42AH Timer, and loops until time out. The subroutine loops to output the next step until all of the Stored Time Constants have been used. The program returns to the Carriage Stepper Motor Drive subroutine and the motor is deselected following the Delay subroutine execution. The Delay subroutine is called to stabilize the stepper motor before it is deselected. During the DELAY subroutine, the IBF interrupt is enabled and characters are processed. A paperfeed is forced following the carriage stepper motor being deselected.

10. Paper Feed Stepper Motor Drive

The paper feed stepper motor subroutine outputs a predefined number of step signals to advance the paper, in one line increments, for the required number of lines. The number of step signals per line increment is a function of the defined number of lines per inch, given the distance the paper moves in one step. Figure 16 lists three step (or pulse) count and line spacing configurations, as well as the distance the paper moves in one step. Standard 6 lines per inch spacing has been implemented in this Application Note (Appendix B details how variable line spacing could be implemented). Flow Chart 11 illustrates the Paper Feed subroutine.



Flow Chart No. 11. Paper Feed Stepper Motor Drive

The number of lines the paper is to be moved is called the "Line Count." The Line Count defaults to one unless the Formfeed flag is set, or the total number of lines previously moved equals a full page. The default total lines per page for this application is 66. When the total number of lines moved equals 66, the paper is moved to the top of the next page. The Top-of-Page is set at power-on or reset.

If the Formfeed flag has been set in the Character Buffer Fill subroutine, the software calculates the number of lines needed for a top of next page paper feed. The resulting line count is loaded in the Line Count Register. The Paper Feed subroutine loops on the line count until done and then returns to the main program body.

Once the Paper Feed subroutine is complete, the software loops to test the End of File (EOF) Flag (see Flow Chart 1). If EOF is set, the print head assembly is moved to the HOME position, the program again enters the External Status Switch Test subroutine, and begins polling the external status switches. If EOF is not set, the program directly calls the External Status Switch Check subroutine, and the program repeats for the next line.

CONCLUSION

Although the full speed, 12 MHz, of the UPI-42AH was used, the actual speed required is approximately 8–9 MHz. 1400 bytes of the available 2k bytes of Program Memory were used; 500 bytes for the 95 character ASCII code dot pattern look-up table, 900 bytes for operational software. This means that the UPI-42AH has excess processing power and memory space for implementing the additional functions such as those listed below and discussed in Appendix B.

Special Characters or Symbols
Lower Case Descenders
Inline Control Codes
Different Character Formats
Variable Line Spacing

The software developed for this Application Note was not fully optimized and could be further packed by combining functions. This would require creating another status register, which could also serve to implement some of the features listed above. Since the full 16 byte stack is not used for subroutine nesting, there are 6–8 bytes of Program Stack Data Memory that could be used for this purpose. In several places, extra code was added for clarity of the Application Note. For example, each status byte flag is set with a separate instruction, using a equate label, rather than setting several flags simultaneously at the same point in the code.

This Application Note has demonstrated that the UPI-42AH is easily capable of independently controlling a complex peripheral device requiring real time event monitoring. The moderate size of the program required to implement this application attests to the effectiveness of the UPI-42AH for peripheral control.

APPENDIX A
SOFTWARE LISTING

```
                1 $MOD42  TITLE('UPI 42 APP NOTE');
                2 $MACROFILE NOSYMBOLS NOGEN DEBUG
                3
                4 $INCLUDE(:F1:ANECD.OV1)
=               5 ; PG
=               6
=               7 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
=               8 ;                 Complex Peripheral Control With the UPI-42
=               9 ;
=              10 ;                 Intel Corporation
=              11 ;                 3065 Bowers Avenue
=              12 ;                 Santa Clara, Ca. 95051
=              13 ;
=              14 ;
=              15 ;                 Written By      Christopher Scott
=              16 ;
=              17 ;
=              18 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
=              19 ;
=              20 ;                 Notes and Comments
=              21 ;
=              22 ;       Three Assembly Language files comprise the full Application
=              23 ;       Note source code:
=              24 ;
=              25 ;         1. ANECD.OV1     App Note Equates, Constants, Declarations . Overlay
=              26 ;
=              27 ;         2. 42ANC.SRC     UPI-42 App Note Code Source
=              28 ;
=              29 ;         3. CHRTBL.OV1    Character Table . Overlay  (Character Lookup Tables)
=              30 ;
=              31 ;
=              32 ;
=              33 ;
=              34 ; PG
=              35 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
=              36 ;                 Equates, Constants and System Definitions
=              37 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
=              38 ;
=              39 ;       Data & Program Memory Allocations
=              40 ;
=              41 ;       Program Memory
=              42 ;
=              43 ;       Page No.   Hex Addr      Description
=              44 ;       ------------------------------------------------------------------------
=              45 ;
=              46 ;       Page 7     1792-2047     Char to Dot pattern lookup table
=              47 ;                                  Page 2:  ASCII 50H-7FH (N-~)
=              48 ;       Page 6     1536-1791     Char to Dot pattern lookup table
=              49 ;                                  Page 1:  ASCII 20H-4FH (sp-M)
=              50 ;       Page 5     1280-1535     Misc called routines:
=              51 ;                                  InitAl/AllOff
=              52 ;                                  Clear Data Memory
=              53 ;                                  CR Home
=              54 ;                                  Char Print Test -  load Ascii char codes
=              55 ;                                  Initialize CR Stpr Mtr
=              56 ;                                  Delay: short/long/very long
=              57 ;                                  Stpr Mtr deselect
=              58 ;       Page 4     1024-1279     PaperFeed Stpr Mtr Init and Drive
=              59 ;       Page 3      768-1023     Stpr Mtr Phase LookUp Table - Indexed
=              60 ;                                Character Translation and processing
=              61 ;                                PrintHead firing
=              62 ;       Page 2      512-767      Stpr Mtr Accel. Time calc. and memorization
=              63 ;                                Stpr Mtr Deceleration
=              64 ;       Page 1      256-511      SMDriv (FAccel/RAccel) - Forward & Reverse
=              65 ;                                  Stpr Mtr acceleration & drive
=              66 ;       Page 0        0-255      Initialization Jmp-on-Reset
=              67 ;                                Program Body - all calls
=              68 ;                                Character Input test and Char Buffer fill loop
=              69 ;                                Interrupt service routines
=              70
```

230795-29

```
                =  71 ; PG
                =  72 ; ------------------------------------------------------------------------------
                =  73 ;       Data Memory
                =  74 ; ------------------------------------------------------------------------------
                =  75 ;         Dec.        Hex           Description
                =  76 ;        -------     -------        ------------------------------------------
                =  77 ;   TOP
                =  78 ;   ---------------------
                =  79 ;      48-127      2F-7FH          80 Character Line Buffer
                =  80 ;       37-47      25-2EH          Stpr Mtr Accel/Decel time, memorization
                =  81 ;          36         24H          Unused
                =  82 ;          35         23H          Char Print test ASCII code start tmp store
                =  83 ;          34         22H          LF SM last Phz Inderect Addr psuedo reg
                =  84 ;          33         21H          CR SM Forward/Reverse last Phz psuedo reg
                =  85 ;          32         20H          Psuedo Reg:  Last Phase of stpr mtr not
                =  86 ;                                              being driven
                =  87 ;       24-31      18-1FH          Register Bank 1:  Character Handling
                =  88 ;        8-23       8-17H          8 Level Stack
                =  89 ;        0-7        0-07H          Register Bank 0:  Stpr Mtr F/R Accel/Drive
                =  90 ;   ---------------------
                =  91 ;   BOTTOM
                =  92
                =  93 ;       Data Memory Equates:
                =  94 ; ------------------------------------------------------------------------------
                =  95
0050            =  96 CHBFSZ  EQU     50H             ;char buffer size 0-79 = 80
00D9            =  97 H1fCpl  Equ     0D9H            ;Cpl(1/2 CbBfSz) => cpl of 27H = 0D9H
                =  98
007F            =  99 FCBfSt  Equ     7fH             ;start of char buffer
0080            = 100 FCBfIS  Equ     80H             ;init CB strt-allows xtra Dec by 1
002F            = 101 RCBfIS  Equ     2FH             ;init CB strt-allows xtra Inc by 1
0051            = 102 ChBfIS  Equ     81              ;load char cnt reg w/char bufr Init Size
                = 103
002F            = 104 ENDBUF  EQU     2FH             ;END OF CHAR BUFFER
0008            = 105 ASBfSz  EQU     08H             ;Accelerate stpr mtr buf count
000A            = 106 DSBfSz  Equ     0AH             ;Decelerate stpr mtr buf count
002F            = 107 SMBFST  EQU     2FH             ;STPR MTR BUFFER START
0025            = 108 SMBEnd  Equ     25H             ;Stpr Mtr Data Memory Address end
007F            = 109 DMTop   Equ     7FH             ;Data Memory Top
005D            = 110 DMSize  Equ     93              ;Data Memory Size (less two working reg's)
                = 111
0020            = 112 LastPh  Equ     20H             ;last phz psuedo reg addr
0021            = 113 CPSAdr  EQU     21H             ;CR phz psuedo reg
0022            = 114 LPSAdr  Equ     22H             ;LF phz psuedo reg
0023            = 115 PTAscS  Equ     23H             ;Char Print Test code start tmp store
                = 116
                = 117 ; PG
                = 118 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 119 ;       Register allocation
                = 120 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 121
                = 122 ;         All Indirect Data Memory Addressing via @Rn Inst must use
                = 123 ;           only registers 0 & 1 of either register bank.  Any other will
                = 124 ;           be rejected by the Assembler
                = 125 ;         Last character in lable indicates Register Bank referenced
                = 126
                = 127 ;       Register Bank 0
                = 128 ; ------------------------------------------------------------------------------
0000            = 129 TmpR0O  Equ     R0              ;RB0 Temporary Register
0001            = 130 TStrR0  EQU     R1              ;Store Time Register RB0
0002            = 131 GStR20  EQU     R2              ;General Status Register RB0
0003            = 132 PhzR30  EQU     R3              ;Stpr Mtr Phase Register RB0
0004            = 133 CntR40  Equ     R4              ;Count Reg. Phase count-Stpr Mtr loops
                = 134                                 ;  Accel/Decel Count
0005            = 135 TConR0  Equ     R5              ;Time constant reg RB0
0006            = 136 LnCtR0  Equ     R6              ;Line count
                = 137
0007            = 138 OpnR7O  EQU     R7              ;available
                = 139
                = 140 ;       Register Bank 0 Data Memory Address
                = 141 ; ------------------------------------------------------------------------------
```

230795-30

```
0000            = 142 TmpA00  Equ     00H             ;Temporary Register DM address
0001            = 143 TStrA0  EQU     01H             ;Time Store Register  DM address
0002            = 144 GStRAd  Equ     02H             ;RBO Char Status Reg  DM address
0003            = 145 PhzA20  EQU     03H             ;Stpr Mtr Phase Register DM address
0004            = 146 CntRA0  Equ     04H             ;Count Reg. Phase count-Stpr Mtr loops
                = 147                                 ;  Accel/Decel Count DM address
0005            = 148 TConA0  Equ     05H             ;Time constant reg DM address
0006            = 149 LnCtA0  Equ     06H             ;Line Count Register DM address
                = 150
0007            = 151 OpnA70  EQU     07H             ;available
                = 152
                = 153
                = 154 ; PG
                = 155 ; ------------------------------------------------------------------------
                = 156 ;       RBO  Status Byte Bit Definition
                = 157 ; ------------------------------------------------------------------------
                = 158
                = 159 ;       Bit             Definition
                = 160 ;       -----           ----------------------------------------------
                = 161 ;       7               Stpr Mtr Direction:  L-to-R = 1, R-to-L = 0
                = 162 ;       6               1 = Sink / 0 = Not Sinked, Print Head Init and Fire
                = 163 ;       5               Stpr Mtr at speed and CR not left of Home
                = 164 ;       4               Accel/Decel Init, 1 = Done / 0 = Not Done
                = 165
                = 166 ;       3               1 = FailSafe / 0 = Constant, Time Window
                = 167 ;       2               1 = Form Feed / 0 = Line Feed
                = 168 ;       1               1 = Do Not Print / 0 = Print
                = 169 ;       0               FAccel/DAccel drive Ready = 1/NotRdy = 0 (exit
                = 170 ;                         drive & decel stpr mtr)
                = 171
                = 172 ;       Bit Masks:      RBO
                = 173 ;       Stepper Motor control bit masks function on GStR10
                = 174 ; ------------------------------------------------------------------
                = 175
                = 176 LRPrnt  Equ     80H             ;Left to Right Printing (ORL)
                = 177 RLPrnt  Equ     7FH             ;Right to Left Printing (ANL)
                = 178 SnkSet  Equ     40H             ;Ready Print flag
                = 179 ClrSnk  Equ     0BFH            ;clear Ready to Print Bit
                = 180 AtSpdF  Equ     20H             ;Stpr Mtr at constant speed
                = 181 NAtSpd  Equ     0DFH            ;Stpr Mtr Not at speed
                = 182 ADIntD  Equ     10H             ;Accel/Decel Init Done
                = 183 ADIntN  Equ     0EFH            ;Accel/Decel Init Not Done
                = 184
                = 185 FsCTm   Equ     08H             ;FailSafe/Constant Time
                = 186 ClrFsC  Equ     0F7H            ;Clear FailSafe/Const time flag
                = 187 FrmFd   Equ     04H             ;do formfeed
                = 188 LineFd  Equ     0FBH            ;do line feed
                = 189 DoNotP  Equ     02H             ;set Do Not Print Stat bit
                = 190 OkPrnt  Equ     0FDH            ;Reset - Ok to Print
                = 191 Ready   Equ     01H             ;Ready drive Stpr Mtr
                = 192 NotRdy  Equ     0FEH            ;Not Ready exit Stpr Mtr drive
                = 193
                = 194
                = 195 ; PG
                = 196 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 197 ;       Register allocation    (cont)
                = 198 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 199
                = 200
                = 201 ;       Register Bank 1
                = 202 ; ------------------------------------------------------------------
0000            = 203 TmpR10  Equ     R0
0001            = 204 CAdrR1  EQU     R1              ;char data memory addr register
0002            = 205 ChStR1  EQU     R2              ;char processing status byte register
0003            = 206 CDtCR1  EQU     R3              ;Char Dot count register
0004            = 207 CDotR1  Equ     R4              ;Char dot temp storage register
0005            = 208 CCntR1  Equ     R5              ;Char count temp register
0006            = 209 StrCR1  EQU     R6              ;Store Char Register
                = 210
0007            = 211 OpnR71  EQU     R7              ;Available
                = 212
                = 213 ;       Register Bank 1 Data Memory Address
                = 214 ; ------------------------------------------------------------------
```

230795-31

```
0018            = 215 TmpA1O  Equ     24              ;temporary/scratch register
0019            = 216 ChARR1  EQU     25              ;char data memory addr register
001A            = 217 ChStAd  Equ     26              ;RB1 Char Status Reg address
001B            = 218 CDtCA1  EQU     27              ;Char Dot count register
001C            = 219 CDotA1  Equ     28              ;Char dot temp storage register
001D            = 220 CCntA1  Equ     29              ;Char count temp register
001E            = 221 StrCA1  EQU     30              ;Store Char Register
                = 222
001F            = 223 OpnA71  EQU     31              ;Available
                = 224
                = 225
                = 226 ; PG
                = 227 ; ---------------------------------------------------------------------
                = 228 ;       RB1  Status Byte Bit Definition
                = 229 ; ---------------------------------------------------------------------
                = 230
                = 231 ;       Bit             Definition
                = 232 ;       -----           ----------------------------------------
                = 233 ;       7               Stpr Mtr Direction:  L-to-R = 1, R-to-L = 0
                = 234 ;       6               Char Init, 1 =  Done / 0 = Not Done
                = 235 ;       5               Char Lookup Table Page: 1 = Pg1, 0 = Pg2
                = 236 ;       4               1 = Test / 0 = Normal char print/input
                = 237
                = 238 ;       3               1 = EOF / 0 = Not EOF
                = 239 ;       2               Full = 1/Not Full = 0, Line in Char Buffer
                = 240 ;       1               1 = CR/(LF) / 0 = Not CR/(LF)
                = 241 ;       0               1 = Init / 0 = Do Not Init, CB registers done
                = 242
                = 243 ;       Bit Masks:      RB1
                = 244 ;       Character printing bit masks function on ChStR1
                = 245 ; ---------------------------------------------------------------------
0080            = 246 ChrPrn  Equ     80H             ;Stpr Mtr Direction:  L-to-R = 1
007F            = 247 ClrCPr  Equ     7FH             ;Stpr Mtr Direction:  R-to-L = 0
0040            = 248 ChIntD  Equ     040H            ;Set Char Init Done
00BF            = 249 CIntND  Equ     0BFH            ;Reset Char Init Not Done
0020            = 250 ChOnP1  Equ     20H             ;Page 1 char, set rentry bit (ORL)
00DF            = 251 ChOnP2  Equ     0DFH            ;Page 2 char, reset rentry bit (ANL)
0010            = 252 TstPrn  Equ     10H             ;Char print test
00EF            = 253 NrmPrn  Equ     0EFH            ;Normal char input
                = 254
0008            = 255 EOF     Equ     08H             ;set EOF Flag
00F7            = 256 ClrEOF  Equ     0F7H            ;clear EOF flag - Not EOF
0004            = 257 CRLF    Equ     04H             ;CR/LF
00FB            = 258 ClrCR   Equ     0FBH            ;Clear CR/LF
0002            = 259 CBFLn   Equ     02H             ;Full Line in Char Buffer
00FD            = 260 NCBFLn  Equ     0FDH            ;Not Full Line in Char Buffer
0001            = 261 IntCBR  Equ     01H             ;Init of CB registers done
00FE            = 262 ClICBR  Equ     0FEH            ;Init of CB registers not done
                = 263
                = 264
                = 265 ; PG
                = 266 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 267 ;                 Equates  (cont)
                = 268 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 269
                = 270 ;       Misc
                = 271 ; ---------------------------------------------------------------
0004            = 272 RLPShf  Equ     04H             ;R-to-L print lookup table addr shift
                = 273
0020            = 274 Ascii   Equ     20H             ;hex nmbr of first Ascii Char
007F            = 275 AscLst  Equ     7FH             ;hex nmbr of last Ascii Char
                = 276
00F3            = 277 CRCpl   Equ     0F3H            ;ASCII control code 2's complement
00F6            = 278 LFCpl   Equ     0F6H            ;  "
00F4            = 279 FFCpl   Equ     0F4H            ;  "
00E5            = 280 EscCpl  Equ     0E5H            ;  "
00E0            = 281 AscCpl  Equ     0E0H            ;  "
00C8            = 282 FTCpl   Equ     0C8H            ;  "
000D            = 283 CR      Equ     0DH             ;Ascii code (hex)
0020            = 284 Space   Equ     20H             ;Ascii code (hex)
                = 285
0081            = 286 LAsEnd  Equ     81H             ;Ascii End 2's cpl - test line start
0082            = 287 PAsEnd  Equ     82H             ;Ascii End 2's cpl - within line print
007F            = 288 AscStp  Equ     7fH             ;Ascii mask, strip off MSB
0042            = 289 PgLnCt  Equ     66              ;Page Line Count: Default = 66
00C4            = 290 PgLCpl  Equ     0C4H            ;Printed lines per page test
001B            = 291 EOFCpl  Equ     1BH             ;EOF ascii code cpl
                = 292
                = 293 ;       Loop count values
                = 294 ; ---------------------------------------------------------------
```

230795-32

```
0006            = 295 NDtCCt  Equ     06H             ;Normal Dot Column Count
000A            = 296 EDtCCt  Equ     0AH             ;Expanded Dot Column Count
0004            = 297 PHCnt1  EQU     04H             ;NUMBER OF SM PHASES ON INIT
                = 298
0004            = 299 ILFCnt  Equ     04              ;Init LF step/phz count
0024            = 300 LPI6p6  Equ     36              ;Lines Per Inch 6.6
001B            = 301 LPI8p8  Equ     27              ;Lines Per Inch 8.8
0018            = 302 LPI10   Equ     24              ;Lines Per Inch 10
                = 303
0001            = 304 LineCt  Equ     01              ;linefeed count
0042            = 305 FmFdCt  Equ     66              ;lines per formfeed count
0003            = 306 Status  EQU     03H             ;SEE BELOW FOR STATUS BYTE DEF.
                = 307                                 ;  TEST:  SET FOR CR STPR MTR CONTROL
                = 308
                = 309
                = 310 ; PG

                = 311 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 312 ;       TIMER VALUES - UPI Timer/Counter is UP Counter
                = 313 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 314 ;       12 MHz Clk timings
                = 315 ; -------------------------------------------------------------
0080            = 316 DLYCL   EQU     80H             ;DELAY COUNT Long
0030            = 317 DLYCS   EQU     30H             ;DELAY COUNT Short
00CC            = 318 DlyTim  EQU     256-52          ;TIME DELAY constant ~2.0mS
0000            = 319 FailTm  EQU     256-256         ;FailSafe TIME = ~17.0mS
00CC            = 320 CrTmr1  EQU     256-52          ;CR Stpr Mtr Phase TIME = ~2.08mS
00BA            = 321 CrTmr2  EQU     256-70          ;CR Stpr Mtr Phase TIME = ~2.40mS
0092            = 322 CrTmr3  EQU     256-110         ;CR Stpr Mtr Phase TIME = ~4.16mS
00C0            = 323 IntTm2  EQU     256-64          ;Init Stpr Mtr Phase TIME = ~2.40mS
0098            = 324 LFTMR1  EQU     256-104         ;LF Stpr Mtr Phase TIME = ~4.16mS
                = 325
                = 326 ;       I/O port bit masks
00DF            = 327 NotBsy  Equ     0DFH            ;Not Busy
0020            = 328 Busy    Equ     20H             ;Busy
00EF            = 329 Ack     Equ     0EFH            ;Ack
0010            = 330 ReSAck  Equ     10H             ;ReSet Ack
                = 331
                = 332 ;       Misc bit Masks
000C            = 333 StrpLF  Equ     0CH             ;Strip off all bits but LF Stpr Mtr
0003            = 334 StrpCR  Equ     03H             ;Strip off all bits but CR Stpr Mtr
                = 335
                = 336 ;       Print Head fires on low going edge of Trigger
                = 337 ;         bit #9 in dot column is masked off, always;  P2, bit 6
0040            = 338 PTRGLO  EQU     40H             ;PH TRIGGER BIT - LOW
00C0            = 339 PTRGHI  EQU     0C0H            ;PH TRIGGER BIT - HIGH
                = 340
                = 341 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 342 ;       Stepper Motor Phase State Equates
                = 343 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 344
                = 345 ;       Stepper Motor Phase Shift Index Offset Offset
0000            = 346 FStCRP  EQU     00H             ;F CR stpr mtr phase data start addr
0003            = 347 RStCRP  EQU     03H             ;R CR stpr mtr phase data start addr
0008            = 348 STLFF   Equ     08H             ;Paper feed stpr mtr phase data start addr
                = 349
                = 350 ;       CARRIAGE STEPPER MOTOR PHASE EQUATES
                = 351 ;       Forward (1 thru 4)  &  Reverse (4 thru 1) :
0001            = 352 CRMFP1  EQU     01B             ;CR STPR MTR PHASE 1
0003            = 353 CRMFP2  EQU     11B             ;CR STPR MTR PHASE 2
0002            = 354 CRMFP3  EQU     10B             ;CR STPR MTR PHASE 3
0000            = 355 CRMFP4  EQU     00B             ;CR STPR MTR PHASE 4
                = 356
                = 357         ;LINE FEED STEPPER MOTOR PHASE EQUATES
                = 358 ;       Forward:
0004            = 359 LFMFP1  EQU     0100B           ;LF STPR MTR PHASE 1
000C            = 360 LFMFP2  EQU     1100B           ;LF STPR MTR PHASE 2
0008            = 361 LFMFP3  EQU     1000B           ;LF STPR MTR PHASE 3
0000            = 362 LFMFP4  EQU     0000B           ;LF STPR MTR PHASE 4
                = 363
                = 364 ; PG
```

230795-33

```
                    = 365 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    = 366 ;         STEPPER MOTOR SELECT & CONTROL [CURRENT LIMITING]
                    = 367 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    = 368
                    = 369 ;         PORT BIT ASSIGNMENT:
                    = 370
                    = 371 ;                 \ \ \
                    = 372 ;                 S S S -
                    = 373 ;                 L C C
                    = 374 ;                 F R R
                    = 375 ;                   8 1
                    = 376 ;                   0 3
                    = 377 ;                     2
                    = 378 ;         ------------------
                    = 379 ;                 5 5 5 5
                    = 380 ;                 3 2 1 0
                    = 381 ;         ------------------
                    = 382 ;  CODING:
                    = 383 ;         SLF     0 1 1 0         06H
                    = 384 ;         SCR80   1 0 0 0         0AH
                    = 385 ;         SCR132  1 1 0 0         0CH
                    = 386 ;         SMOFF   1 1 1 0         0EH
                    = 387 ;                 W/SCR80 & SCR132 '0' [BOTH SELECTED]
                    = 388 ;                   DEFAULT IS TO 80 COL.
                    = 389 ;                   [DO NOT KNOW WHETHER SCR80='0' WILL
                    = 390 ;                   SELECT 80 COL ONLY] - REQUIRES TEST.
                    = 391
0008                = 392 SCR80   EQU     08H             ;SELECT CR STPR MTR - 80 COL
                    = 393                                 ;  w/LF STPR MTR OFF
000C                = 394 SCR132  EQU     0CH             ;SELECT CR STPR MTR - 132 COL
                    = 395                                 ;  w/LF STPR MTR OFF
0006                = 396 SLF     EQU     06H             ;SELECT LF STPR MTR ON
                    = 397                                 ;  w/CR STPR MTR OFF
000E                = 398 SMOFF   EQU     0EH             ;SELECT CR & LF STPR MTR OFF
                    399
                    400
                    401 ; PG
                    402 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    403 ;                   MAIN PROGRAM BODY
                    404 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    405
                    406 ;         Power On / Reset Program Entry
                    407
                    408 ;         PROGRAM START
                    409
0000                410           Org     00H
                    411
0000 040B           412 START:    JMP     RESET
                    413
                    414 ;         INPUT BUFFER FULL INTERRUPT CALL ENTRY AND VECTOR
0003                415           ORG     03H
0003 1425           416 IBFIV:    Call    IBFIS
0005 93             417           RETR
                    418 ;         TIMER OVERFLOW INTERRUPT CALL ENTRY AND VECTOR
0007                419           ORG     07H
0007 1429           420 TMRIV:    Call    TMRIS
0009 C5             421           SEL     RB0
000A 83             422           Ret
                    423
                    424 ;                   INITIALIZATION
                    425
000B 15             426 ReseT:    Dis     I
000C 35             427           Dis     TCntI
000D B40F           428           Call    InitAl          ;set all critical outputs inactive
000F B42F           429           Call    ClrDM           ;clear all data memory - 93H to 7FH
                    430                                   ;  do not clear RB0, RB1 or Stack
0011 B44B           431           Call    InitCR          ;CALL CR SM POWER ON INIT
0013 9400           432           Call    InitLF          ;CALL LF SM POWER ON INIT
                    433
                    434 ;                   MAIN PROGRAM LOOP
                    435 ;         All program segments are called from here
                    436
0015 B422           437 Home:     Call    CRHome          ;Call Home CR routine -
                    438                                   ;  fixes logical and physical CR Home
0017 B400           439           Call    Defalt          ;set default register values
0019 142C           440 CBInpt:   Call    ESCBfF          ;Stat Switch / CB Input Service Test
                    441                                   ; test for:  CB full/fill, LF, FF,
                    442                                   ;            Char Prnt Test
```

230795-34

```
001B 3400          443 Repeat:  Call    SMDriv          ;Call Forward Stpr Mtr Drive
001D 940D          444          Call    LFDriv          ;Call Linefeed Stpr Mtr Drive
001F D5            445          SEL     RB1
0020 FA            446          Mov     A,ChStR1        ;get the Char Status Register RB1
0021 7215          447          JB3     Home            ;jump to CR SM Home if EOF bit set
0023 0419          448          Jmp     CBInpt          ;loop to Char Buffer Input test
                   449
                   450 ; PG
                   451 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   452 ;                   Interrupt Service Routine
                   453 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   454
                   455 ; -----------------------------------------------------------------------------
                   456 ;                   Input Buffer Full Interrupt Service Routine
                   457 ; -----------------------------------------------------------------------------
                   458
                   459 IBFIS:
                   460 ; -----------------------------------------------------------------------------
                   461 ;          Acknowledge Char input and set Hold/Busy Active
0025 8A20          462          ORL     P2,#Busy        ;get & set DBB ACK/Busy Bits
0027 15            463          Dis     I               ;disable IBF interrupts
0028 83            464          Ret
                   465
                   466 ; -----------------------------------------------------------------------------
                   467 ;                   Timer / Counter Interrupt Service Routine
                   468 ; -----------------------------------------------------------------------------
                   469 ;          ITF interrupt service routine disables all intr during
                   470 ;            stpr mtr phase shifting
0029 15            471 TMRIS:   Dis     I               ;disable IBF interrupts
002A 35            472          Dis     TCntI           ;dicable ITF interrupts
002B 83            473          Ret
                   474
                   475 ; PG
                   476 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   477 ;          External Status Switch Check/Char. Buffer Fill
                   478 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   479 ESCBfF:                          ;Prep for normal character handling/input
002C D5            480          SEL     RB1
002D FA            481          Mov     A,ChStR1        ;get the character stat reg byte
002E 53EF          482          ANL     A,#NrmPrn       ;set normal character input
0030 AA            483          Mov     ChStR1,A        ;store the stat byte
0031 C5            484          SEL     RB0
                   485
                   486 ;          Test External Status Port
0032 0F            487          MovD    A,P7            ;get the stat switch port bits
0033 123D          488          JB0     FormFd          ;  service Formfeed
0035 3245          489          JB1     LinFd           ;  service Linefeed
0037 5249          490          JB2     ChrTst          ;  service Character TEST
0039 725E          491          JB3     OnLine          ;  service Char Buffer Check/Fill
003B 042C          492          Jmp     ESCBfF          ;Loop
                   493 ; -----------------------------------------------------------------------------
003D FA            494 FormFd:  Mov     A,GStR20        ;get the status byte
003E 4304          495          ORL     A,#FrmFd        ;set the formfeed stat flag
0040 AA            496          Mov     GStR20,A        ;store trhe status byte
0041 940D          497          Call    LfDriv          ;do a formfeed
0043 042C          498          Jmp     ESCBfF
                   499 ; -----------------------------------------------------------------------------
0045 940D          500 LinFd:   Call    LfDriv          ;do a line drive
0047 042C          501          Jmp     ESCBfF
                   502 ; -----------------------------------------------------------------------------
0049 D5            503 ChrTst:  SEL     RB1
004A FA            504          Mov     A,ChStR1        ;get the character stat reg byte
004B 4310          505          ORL     A,#TstPrn       ;set character test flag
004D AA            506          Mov     ChStR1,A        ;store the stat byte
004E B823          507          Mov     TmpR10,#PTAscS  ;load the psuedo Ascii code tmp reg addr
0050 F0            508          Mov     A,@TmpR10       ;get the inc'd ascii code
0051 0381          509          ADD     A,#LAsEnd       ;test for code end
0053 9657          510          JNZ     AscCLd          ;if not code end jmp to load
                   511                                  ;if end restart ascii at begining
0055 B020          512          Mov     @TmpR10,#Ascii  ;store the ascii code start
0057 F0            513 AscCLd:  Mov     A,@TmpR10       ;get the ascii code again
0058 AF            514          Mov     OpnR71,A        ;place in the empty register
0059 10            515          Inc     @TmpR10         ;Inc start ASCII char in data memory
005A B439          516          Call    PrnTst          ;call the DM load procedure
005C C5            517          SEL     RB0             ;reselect reg bank 0
005D 83            518          Ret
                   519 ; -----------------------------------------------------------------------------
```

230795-35

```
005E D5          520 OnLine:  SEL     RB1             ;select char buffer registers
005F 05          521          EN      I               ;enable interrupts
0060 FA          522 CBfCk1:  Mov     A,ChStR1        ;get the Char Stat Byte
0061 3267        523          JB1     CBCkEx          ;if Chr Buf has full line exit
0063 146D        524 IBfCk:   Call    CBFill          ;read a char into Char Buffer
0065 0460        525          Jmp     CBFCk1          ;loop to Char Buf Ful test
0067 C5          526 CBCkEx:  SEL     RB0
0068 83          527          Ret
                 528
                 529 ; PG
                 530 ; -----------------------------------------------------------------------
                 531 ;       Character Input
                 532 ; -----------------------------------------------------------------------
                 533 ;       Input Buffer Full service routine:  test for Char buffer full-exit
                 534 ;         else load char into char buffer
0069 D5          535 IBFSrv:  SEL     RB1
006A FA          536          Mov     A,ChStR1        ;get the RBO stat byte
006B 32EC        537          JB1     CBFull          ;if Do Not Print Bit Set - EXIT
006D 527C        538 CBFill:  JB2     CBPad           ;test for CB padding flag
                 539                                  ;  if not pad enable char input
                 540                                  ;  tell the host to send char's
006F 05          541          EN      I
0070 D6EC        542          JNIBF   CBF1Ex
                 543 ; -----------------------------------------------------------------------
                 544 ;       Acknowledge Char input and set Hold/Busy Active
0072 FA          545          Mov     A,ChStR1        ;get the RB1 Char Stat Byte
0073 127C        546          JBO     SkpInt          ;test for CB has been Initialized
                 547 ; -----------------------------------------------------------------------
                 548 ;       Init of all Char handling registers
0075 4301        549          ORL     A,#IntCBR       ;set CB Reg skip Initialization stat bit
0077 AA          550          Mov     ChStR1,A        ;save the altered stat byte
0078 B97F        551          Mov     CAdrR1,#FCBfSt  ;load char reg w/char bufr strt
007A BD50        552          Mov     CCntR1,#ChBfSz  ;load char cnt reg w/char bufr size
                 553 CBPad:
007C ED86        554 SkpInt:  DJNZ    CCntR1,LdChar   ;DECREMENT BUFFER SIZE
007E FA          555          Mov     A,ChStR1        ;get the status byte
007F 4302        556          ORL     A,#CBFLn        ;set Char Buffer Full Line stat bit
0081 53FB        557          ANL     A,#ClrCr        ;clear the CR/(LF) stat bit
0083 53FE        558          ANL     A,#ClICBR       ;reset CB Init bit: init CB reg on entry
0085 AA          559          Mov     ChStR1,A        ;store the status byte
0086 FA          560 LdChar:  Mov     A,ChStR1        ;get the status byte
0087 52E1        561          JB2     CBPad1          ;CB not full but CR/LF previously
                 562                                  ;  received so pad CB
0089 9AEF        563          ANL     P2,#Ack         ;output DBB Ack low
008B 22          564          In      A,DBB           ;read the Char
008C 537F        565          ANL     A,#AscStp       ;strip off MSB
008E AB          566          Mov     TmpR10,A        ;temp save char
008F BA10        567          ORL     P2,#ReSAck      ;output DBB ACK High
                 568 ; -----------------------------------------------------------------------
                 569 ;       test for ASCII printable character
0091 03E0        570          ADD     A,#ASCCp1       ;test for Carriage Return
0093 F697        571          JC      AsciiC          ;jmp to service
0095 049C        572          Jmp     ChrChk
0097 97          573 AsciiC:  Clr     C               ;clear carry flag
0098 F8          574          Mov     A,TmpR10        ;get the char back
0099 A1          575          Mov     @CAdrR1,A       ;load data memory w/Char
009A 04E3        576          Jmp     IBFSrE
                 577 ; -----------------------------------------------------------------------
                 578 ;       test for CR/LF:  if CR/LF Strip off LF and exit setting
                 579 ;         Char Buffer Init Stat bit
009C F8          580 ChrChk:  Mov     A,TmpR10        ;get the char back
009D 03F3        581          ADD     A,#CRCp1        ;test for Carriage Return
009F C6C3        582          JZ      CRChr           ;  if CR go service it
00A1 F8          583          Mov     A,TmpR10        ;get the char back
00A2 031B        584          ADD     A,#EOFCp1       ;test for End Of File
00A4 96AA        585          JNZ     ChrCk1          ;if not EOF jmp to CB Pad
00A6 F8          586          Mov     A,TmpR10        ;if EOF, place it in CB
00A7 A1          587          Mov     @CAdrR1,A       ;load data memory w/CR Char
00A8 04B9        588          Jmp     ExtSet          ;Exit
00AA F8          589 ChrCk1:  Mov     A,TmpR10        ;get the status byte
00AB 03F4        590          ADD     A,#FFCp1        ;test for FormFeed
00AD 96E1        591          JNZ     CBPad1          ;if not FF Pad the CB
00AF C5          592          SEL     RB0
00B0 FA          593          Mov     A,QStR20        ;get the status byte
00B1 4304        594          ORL     A,#FrmFd        ;set the formfeed flag
00B3 AA          595          Mov     QStR20,A        ;store the status byte
00B4 D5          596          SEL     RB1
00B5 FA          597          Mov     A,ChStR1        ;get the status byte
00B6 4304        598          ORL     A,#CRLF         ;set CRLF stat bit:  pad balance of CB
                 599                                  ;  with Spaces until fill
00B8 AA          600          Mov     ChStR1,A        ;store the status byte
00B9 FA          601 ExtSet:  Mov     A,ChStR1        ;get the status byte
```

230795-36

```
00BA 4302          602           ORL     A,#CBFLn        ;set Char Buffer Full Line stat bit
00BC 53FB          603           ANL     A,#ClrCr        ;clear the CR/(LF) stat bit
00BE 53FE          604           ANL     A,#ClICBR       ;reset CB Init bit: init CB reg on entry
00C0 AA            605           Mov     ChStR1,A        ;store the status byte
00C1 04EC          606           Jmp     CBFlEx          ;Exit
                   607 ; ---------------------------------------------------------------------------
                   608 ;         Store CR char read in LF char (assume its always there) and ignor it
00C3 F8            609 CRChr:    Mov     A,TmpR10        ;get the char back
00C4 A1            610           Mov     @CAdrR1,A       ;load data memory w/CR Char
00C5 C5            611           SEL     RB0
00C6 1E            612           INC     LnCtR0          ;inc the line count
00C7 FE            613           Mov     A,LnCtR0        ;get the line count
00C8 03C4          614           Add     A,#PgLCpl       ;test for page feed ln cnt
                   615                                   ;  if LnCt => PgLnCt set formfeed flag
00CA E6D0          616           JNC     NoFmFd          ;if not at end of page skip
00CC FA            617           Mov     A,GStR20        ;get the status byte
00CD 4304          618           ORL     A,#FrmFd        ;set the form feed status flag
00CF AA            619           Mov     GStR20,A        ;save the status byte
00D0 D5            620 NoFmFd:   SEL     RB1
00D1 05            621           En      I               ;enable the IBF service
00D2 9ADF          622           ANL     P2,#NotBsy      ;output a not busy to Host
00D4 D6D4          623 LFTest:   JNIBF   LFTest          ;loop to next char
00D6 9AEF          624           ANL     P2,#Ack         ;output DBB Ack low
00D8 22            625           In      A,DBB           ;get next Char - assume it's a LF
                   626                                   ;  and ignor it (LF is forced upon
                   627                                   ;  detection of CR at print time)
00D9 FA            628 SetPad:   Mov     A,ChStR1        ;get the status byte
00DA 4304          629           ORL     A,#CRLF         ;set CRLF stat bit:  pad balance of CB
                   630                                   ;  with Spaces until fill
00DC AA            631           Mov     ChStR1,A        ;store the status byte
00DD 8A10          632           ORL     P2,#ReSAck      ;output DBB ACK High
00DF 04E3          633           Jmp     IBfSrE          ;jmp to addr step & exit
                   634 ; ---------------------------------------------------------------------------
                   635 ;         fill Char Buffer with space
00E1 B120          636 CBPad1:   Mov     @CAdrR1,#Space  ;load data memory w/Char
                   637 ; ---------------------------------------------------------------------------
                   638 ;         step the char address test for CB full &/or pad
00E3 C9            639 IBFSrE:   DEC     CAdrR1          ;Decrement dat memory location
00E4 FA            640           Mov     A,ChStR1        ;get the status byte
00E5 32EC          641           JB1     CBFull          ;test for CB Full
00E7 52EC          642           JB2     CBFlEx          ;test for CB pad - exit w/Busy set
                   643 ; ---------------------------------------------------------------------------
                   644 ;         Set Busy Line Low - Not Busy
00E9 05            645           EN      I
00EA 9ADF          646           ANL     P2,#NotBsy      ;output a not busy to Host
                   647 ; ---------------------------------------------------------------------------
                   648 ;         exit w/ Busy Still set high
                   649 CBFull:
00EC 83            650 CBFlEx:   Ret
                   651
                   652 ; PG
                   653 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   654 ;         L-to-R/R-to-L Carriage Stepper Motor Drive
                   655 ;             and Line Printing
                   656 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   657
0100               658           ORG     100H
                   659
0100 3622          660 SMDriv:   JTO     RAccel          ;if Print Head at left drive right
                   661                                   ;  else drive left
                   662 ; F=======================================================================
                   663 FAccel:                           ;L-to-R Accelerate Stepper Motor
                   664 ;         Set the Forward acceleration/drive Entry status bits
0102 FA            665           Mov     A,GStR20        ;get the status byte
0103 53BF          666           ANL     A,#ClrSnk       ;set not at speed flag = 0
0105 53DF          667           ANL     A,#NAtSpd       ;set Not At Speed flag = 0
0107 4380          668           ORL     A,#LRPrnt       ;set L-to-R prnt stat bit = 1
0109 4301          669           ORL     A,#Ready        ;set stpr mtr ready - Drive On
010B 53EF          670           ANL     A,#ADIntN       ;set A/D Init Not Done
010D AA            671           Mov     GStR20,A        ;store the status byte
010E D5            672 CBRDir:   SEL     RB1
010F FA            673           Mov     A,ChStR1        ;get the Char Stat Reg Data Mem Addr
0110 4380          674           ORL     A,#LRPrnt       ;Set L-to-R print bit
0112 AA            675           Mov     ChStR1,A        ;save the Char Stat byte
0113 C5            676           SEL     RB0
                   677
                   678 ;         Restore the phase register index addresses
0114 B821          679           Mov     TmpR00,#CPSAdr  ;get Phz Storage Addr psuedo reg
0116 F0            680           Mov     A,@TmpR00       ;  get stored CR last phase index addr
0117 AB            681           Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
                   682
```

230795-37

```
                        683 ;       Set up for next phase bit output before entering timing loops
0118 1B                 684         INC     PhzR30          ;STEP PHASE DB ADDRESS
0119 FB                 685         MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
011A 521E               686         JB2     IAFZrP          ;CHK FOR COUNT BIT ROLLOVER
011C 2440               687         JMP     SMDflt          ;skip adr index reset
011E BB00               688 IAFZrP: MOV     PhzR30,#FStCRP  ;ZERO CR SM PHASE REGISTER
0120 2440               689         Jmp     SMDflt
                        690
                        691 ; R=========================================================================
                        692 RAccel:         ;R-to-L Accelerate Stepper Motor
                        693 ; -------------------------------------------------------------------------
                        694 ;       Set the Reverse acceleration/drive Entry status bits
0122 FA                 695         Mov     A,GStR20        ;get the status byte
0123 53BF               696         ANL     A,#ClrSnk       ;clear Print Ready bit
0125 53DF               697         ANL     A,#NAtSpd       ;set Not At Speed flag = 0
0127 537F               698         ANL     A,#RLPrnt       ;set R-to-L prnt status bit
0129 4301               699         ORL     A,#Ready        ;set stpr mtr ready - Drive On
012B 53EF               700         ANL     A,#ADIntN       ;set A/D Init Not Done
012D AA                 701         Mov     GStR20,A        ;store the status byte
012E D5                 702 RCBRDr: SEL     RB1
012F FA                 703         Mov     A,ChStR1        ;get the Char Stat Reg Data Mem Addr
0130 537F               704         ANL     A,#RLPrnt       ;Set R-to-L print bit
0132 AA                 705         Mov     ChStR1,A        ;save the Char Stat byte
0133 C5                 706         SEL     RB0
                        707 ; -------------------------------------------------------------------------
                        708 ;       Restore the  phase register index address
0134 B821               709         Mov     TmpR00,#CPSAdr  ;get Phz Storage Addr psuedo reg
0136 F0                 710         Mov     A,@TmpR00       ;  get stored CR last phase index addr
0137 AB                 711         Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
                        712 ;       Set up for next phase bit output before entering timing loops
0138 CB                 713         Dec     PhzR30          ;STEP PHASE DB ADDRESS
0139 FB                 714         MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
013A 523E               715         JB2     IARZrP          ;CHK FOR COUNT BIT ROLLOVER
013C 2440               716         JMP     SMDflt
013E BB03               717 IARZrP: MOV     PhzR30,#RStCRP  ;ZERO CR SM PHASE REGISTER
                        718
                        719 SMDflt:
                        720 ; -------------------------------------------------------------------------
                        721 ;       for stablization of unused stpr mtr during CR stpr mtr drive,
                        722 ;         store the unused stpr mtr current phase bits
0140 B822               723         Mov     TmpR00,#LPSAdr  ;get the CR phz storeage addr
0142 F0                 724         Mov     A,@TmpR00       ;get the byute stored there
0143 E3                 725         MovP3   A,@A            ;get the phz data byte
0144 B820               726         Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0146 A0                 727         Mov     @TmpR00,A       ;store Last Phase bits - inderect
                        728
                        729 ;       SetUp Stpr Mtr Time Constant
0147 BDBA               730         MOV     TConR0,#CrTmr2  ;Load time constant Reg
                        731
                        732 Select:                         ;Select the Stpr Mtr
0149 2308               733         MOV     A,#SCR80        ;GET CR SM SELECT BITS
014B 3D                 734         MOVD    P5,A            ;SELECT SM [SCR80]
                        735 ; -------------------------------------------------------------------------
                        736 ;       SetUp Stpr Mtr Phase Shift index address register
                        737 ;       Output next phase and init timer to Std Time constant
014C FD                 738 STRTT:  MOV     A,TConR0        ;get time constant from reg
014D 62                 739         MOV     T,A             ;load the timer
014E FB                 740         MOV     A,PhzR30        ;get the phz reg indirect addr index
014F E3                 741         MovP3   A,@A            ;do indirect get of phz bits
                        742
                        743 ; -------------------------------------------------------------------------
                        744 ;       patch together the CR last and LF next phase bits
0150 B820               745         Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0152 40                 746         ORL     A,@TmpR00       ;patch together CR existing & new LF
0153 3C                 747         MOVD    P4,A            ;OUTPUT BITS
0154 55                 748         STRT    T               ;START TIMER
                        749
                        750 ;       At start of timing loop do all Stpr Mtr Accel/Decel or
                        751 ;         Character SetUp overhead
0155 740C               752         Call    ADPTst          ;call Accel/Decel/Print Test
                        753
                        754 ;       Set up for next phase bit output before entering timing loops
                        755 PNRdy1: ;test for forward / reverse phase start indirect index to load
0157 FA                 756         Mov     A,GStR20        ;store stat byte
0158 F264               757         JB7     Ac1F2
                        758
                        759 ; reverse:
                        760 ;       Set up for next phase bit output before entering timing loops
015A CB                 761         Dec     PhzR30          ;STEP PHASE DB ADDRESS
015B FB                 762         MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
015C 5260               763         JB2     ARZroP          ;CHK FOR COUNT BIT ROLLOVER
015E 2462               764         JMP     ARNxtP
0160 BB03               765 ARZroP: MOV     PhzR30,#RStCRP  ;ZERO CR SM PHASE REGISTER
0162 246C               766 ARNxtP: Jmp     ANxtPh
```

230795-38

```
                        767
                        768 ; forward:
                        769 ;       Set up for next phase bit output before entering timing loops
0164 1B                 770 AclF2:  INC     PhzR30          ;STEP PHASE DB ADDRESS
0165 FB                 771         MOV     A,PhzR30        ;CHECK THE PHASE COUNT REQ
0166 526A               772         JB2     AFZroP          ;CHK FOR COUNT BIT ROLLOVER
0168 246C               773         JMP     ANxtPh          ;skip adr index reset
016A BB00               774 AFZroP: MOV     PhzR30,#FStCRP  ;ZERO CR SM PHASE REGISTER
                        775 ANxtPh:
                        776 ; ---------------------------------------------------------------
                        777 ;       stage one timer loop - T occurs before Std timeout
                        778 ;         wait for time out
016C 1682               779 TLOOP2: JTF     FAILSF          ;JMP ON TIME OUT-t DOES NOT OCCUR 1ST
016E 5672               780         JT1     tCHK1           ;IS T HIGH-JMP TO tCHK
0170 246C               781         JMP     TLOOP2          ;LOOP FOR JT1 OR JTF
0172 00                 782 tCHK1:  NOP                     ;delay, then double check T signal
0173 5677               783         JT1     tTruW1          ;JUMP T TEST TRUE-WAIT FOR JTF
0175 246C               784         JMP     TLOOP2
                        785 tTruW1:
                        786 ;       test for Print Ready bit - was Print Head Fire Setup Done?
                        787 ;       insert acceleration time/store time count done/notdone flag bit
0177 FA                 788         Mov     A,GStR20        ;get the status byte - prep for prnt
0178 D27C               789         JB6     RdyPr2          ;if Ready Print bit set call PHFire
017A 247E               790         Jmp     SkpPHF          ;  else skip Print Head Fire
017C 74CA               791 RdyPr2: Call    PHFire          ;print head solenoid fire routine
                        792 PNRdy2:
                        793 SkpPHF:
017E 1698               794 tTruW2: JTF     NXTPHZ          ;JUMP TO SM ERROR
0180 247E               795         JMP     tTruW2          ; LOOP TO TLOOP3
                        796 ; ---------------------------------------------------------------
                        797 ;       Step into failsafe/startup timer setup - T does not
                        798 ;         occurs before Std Time timeout, load failsafe SM protection
                        799 ;         time and wait for failsafe timeout or T.  If T occurs
                        800 ;         output phase immediately after T verify.
0182 2300               801 FAILSF: MOV     A,#FailTm       ;LOAD TIMER W/~15.0mS
0184 62                 802         MOV     T,A             ;  SM PROTECTION TIMEOUT
0185 55                 803         STRT    T               ;START TIMER
                        804 ; ---------------------------------------------------------------
                        805 ;       set the Status bit for Store time test
0186 FA                 806         Mov     A,GStR20        ;get the status byte
0187 4308               807         ORL     A,#FSCTm        ;set Failsafe/constant time flag
0189 AA                 808         Mov     GStR20,A        ;store the status byte
018A 5690               809 TLOOP3: JT1     tCHK2           ;IS T HIGH
018C 16AC               810         JTF     DSLECT          ;IF TIME OUT GO SM ERROR
018E 248A               811         JMP     TLOOP3          ;LOOP UNTIL T HIGH OR T-OUT
0190 00                 812 tCHK2:  NOP                     ;WAIT
0191 5695               813         JT1     StrTm1          ;jump out and store elapsed time
0193 248A               814         JMP     TLOOP3          ;  JMP TO FAILSF LOOP
0195 65                 815 StrTm1: Stop    TCnt            ;stop the failSafe Timer
0196 42                 816         Mov     A,T             ;read the timer
0197 A1                 817         MOV     @TStrR0,A       ;Store the time read in indexed addr
                        818                                 ; - next entry to A/D Memorize Time
                        819                                 ; routine will add time constant to it
                        820
                        821 ;    Test is CR Stpr Mtr Drive is finished prior to next phase output
                        822 ; ---------------------------------------------------------------
                        823 NXTPHZ:
                        824 ;       test for forward / reverse phase start indirect index to load
0198 FA                 825         Mov     A,GStR20        ;store stat byte
0199 F2A7               826         JB7     FDrive
                        827 ; Reverse  --  test for Reverse Stpr Mtr Drive procedure exit
                        828 ;       ALWAYS drive the CR to the left most HOME position
019B 26AC               829         JNTO    EOLn            ;test if home position jmp stop
019D FA                 830         Mov     A,GStR20        ;get the status byte
019E 124C               831         JB0     StrtT           ;test Ready stat bit:
                        832                                 ;  if bit 0 = 1 then Print More
01A0 4302               833         ORL     A,#DoNotP       ;set the do not print flag
01A2 53BF               834         ANL     A,#ClrSnk       ;clear Print Ready bit
01A4 AA                 835         Mov     GStR20,A        ;save the status byte
01A5 244C               836         Jmp     StrtT           ;continue CR SM drive
                        837                                 ;  - only exit is HR
                        838 ; Forward  --  test for Forward Stpr Mtr Drive procedure exit
                        839 FDrive:
01A7 FA                 840         Mov     A,GStR20        ;get the status byte
01A8 124C               841         JB0     StrtT           ;test Ready stat bit:
                        842                                 ;  if bit 0 = 1 then Print More
01AA 24AC               843         Jmp     EOLn            ;  else jmp to End Of Line exit
                        844                                 ;jump to start timer again
                        845 DSLECT:
01AC 5437               846 EOLn:   Call    DeclSM          ;call Sptr Mtr Deceleration
                        847 ; ---------------------------------------------------------------
                        848 ;       test for forward / reverse phase start indirect index to load
01AE FA                 849         Mov     A,GStR20        ;store stat byte
01AF F2B3               850         JB7     FDrvFS          ;jmp to f drive flag set
```

230795-39

```
01B1 53FD          851          ANL     A,#0kPrnt       ;reset print flag - Ok Print
                   852                                  ; only if printing R-to-L
                   853
                   854 ;        update the status byte
01B3 53BF          855 FDrvFS:  ANL     A,#ClrSnk       ;clear Print Ready bit
                   856                                  ;set the Status bit for Store time test
01B5 53DF          857          ANL     A,#NAtSpd       ;Clear At Print Speed Bit
01B7 AA            858          Mov     GStR20,A        ;save the status byte
01B8 83            859          RET
                   860
                   861 ; PG
                   862 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   863 ;        Stepper Motor Accel. Time Storeage
                   864 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   865
0200               866          ORG     200H
                   867                                  ;   Entry has Gen Stat Byte in A
0200 920C          868 ADMmTS:  JB4     DADInt          ;is A/D init done - then jmp
                   869
                   870 ;        1st Entry initializes the A/D Time store working registers
0202 B92F          871          Mov     TStrR0,#SMBfSt  ;Load the Stpr Mtr Buffer Start Addr
0204 BC0B          872          Mov     CntR4O,#ASBfSz  ;Load the Buffer Size
0206 FA            873          Mov     A,GStR20        ;get the status byte
0207 4310          874          ORL     A,#ADIntD       ;set not 1st Accel Entry Flag
0209 AA            875          Mov     GStR20,A        ;store the status byte
020A 4436          876          Jmp     ADExit          ;exit - 1st entry has not generated
                   877                                  ;   a closed time window
                   878
                   879 ;        Step the A/D Store count
020C EC26          880 DADInt:  DJNZ    CntR4O,StorCt   ;dec Times to store count
                   881                                  ;if not O store the count
                   882                                  ;else at end-set done flag
020E FA            883          Mov     A,GStR20        ;get the status byte
020F 4320          884          ORL     A,#AtSpdF       ;set at speed/no more to store flag
0211 AA            885          Mov     GStR20,A        ;store the status byte
                   886
                   887 ;        Initialize Char Print Registers:  if printing enabled
0212 3226          888          JB1     StorCt          ;if Do Not Print stat bit set
                   889                                  ;  Skip the Char register init
                   890
                   891 ;        Initialize all Char Reg's
                   892 ;        Test for L-to-R (forward) or R-to-L (reverse) printing
0214 D5            893          SEL     RB1
0215 FA            894          Mov     A,ChStR1        ;get the status byte
0216 4340          895          ORL     A,#CHIntD       ;set Char Init Done flag - bypass
0218 AA            896          Mov     ChStR1,A        ;save the status byte
0219 F21F          897          JB7     LdCBR1          ;test Chr Stat Byte Returned
                   898                                  ;  if bit 7 = 1 then Print L-to-R
021B B92F          899 LdCBR:   Mov     CAdrR1,#RCBfIS  ;load char reg w/char bufr strt R-to-L
021D 4421          900          Jmp     LdCBR2
                   901
021F B980          902 LdCBR1:  Mov     CAdrR1,#FCBfIS  ;load char reg w/char bufr strt L-to-R
                   903
0221 BD51          904 LdCBR2:  Mov     CCntR1,#ChBfIS  ;load char cnt reg w/char bufr size
0223 BB01          905          Mov     CDtCR1,#01      ;set the chr dot column cnt
0225 C5            906          SEL     RB0
                   907
                   908 ;        Test for t > Tc or t < Tc
0226 722C          909 StorCt:  JB3     FailST          ;test for failsafe time switch
                   910
                   911 ;        t < Tc = store Time Constant in use
0228 FD            912          Mov     A,TConR0        ;Get time constant currently in use
0229 A1            913          Mov     @TStrR0,A       ;Memorize/Store the time - indirect addr
022A 4435          914          Jmp     ADPRet
                   915
                   916 ;        t > Tc = store Time Constant + FailSafe Time Elapsed
                   917 ;        [see Accel/Cnst Speed/Decel WaveForm]
                   918 ;        equation is:     Trd - FailSafe Time = Tx
                   919 ;                     => Trd + Cpl(FailSafe Time) = Tx
                   920 ;                        Tx + Tcnst = T
                   921 ;                        Store/Memorize T
                   922
022C F1            923 FailST:  Mov     A,@TStrR0       ;get the stored time
022D 03C8          924          Add     A,#FTCpl        ;2's cpl add
022F 6D            925          Add     A,TConR0        ;Add: Time stored + Time constant
                   926                                  ;  currently in use
0230 A1            927          Mov     @TStrR0,A       ;Memorize/Store the time
                   928 ;        Reset the Status bit for Store time test
                   929
0231 FA            930          Mov     A,GStR20        ;get the status byte
0232 53F7          931          ANL     A,#ClrFSC       ;reset Failsafe/constant time flag
                   932                                  ;  assumes entry via constant time
```

230795-40

```
0234 AA          933          Mov     GStR20,A          ;store the status byte
0235 C9          934 ADPRet:  Dec     TStrR0            ;step the A/D time data store addr
0236 83          935 ADExit:  Ret
                 936
                 937 ; PG
                 938 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 939 ;       Carriage Stepper Motor Deceleration
                 940 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 941
                 942 DeclSM:
                 943 ;        SetUp the Deceleration registers
0237 B925        944          Mov     TStrR0,#SMBEnd    ;Load the Stpr Mtr Buffer End Addr
0239 BC0A        945          Mov     CntR40,#DSBfSz    ;Load the Buffer Size
023B FB          946          MOV     A,PhzR30          ;get phase index address
023C E3          947          MovP3   A,@A              ;get phase from indexed address
                 948 ;        patch together the CR last and LF next phase bits
023D B820        949          Mov     TmpR00,#LastPh    ;load Last Phz psuedo reg to Temp Reg
023F 40          950          ORL     A,@TmpR00          ;patch together CR existing & new LF
0240 3C          951          MOVD    P4,A              ;OUTPUT BITS
0241 F1          952 StrtTD:  MOV     A,@TStrR0         ;get time from indexed data memory
0242 62          953          MOV     T,A               ;load timer
0243 55          954          STRT    T                 ;START TIMER
0244 19          955          Inc     TStrR0            ;step the Memorized time addr index reg
                 956 ;        test for forward / reverse phase start indirect index to load
0245 FA          957          Mov     A,GStR20          ;store stat byte
0246 F252        958          JB7     DclF2
                 959
                 960 ; reverse:
                 961 ;        Set up for next phase bit output before entering timing loops
0248 CB          962          Dec     PhzR30            ;decrement the phase addr
0249 FB          963          MOV     A,PhzR30          ;Get the phz data addr
024A 524E        964          JB2     DRZroP            ;CHK FOR COUNT BIT ROLLOVER
024C 445A        965          JMP     DNxtPh
024E BB03        966 DRZroP:  MOV     PhzR30,#RStCRP    ;ZERO CR SM PHASE REGISTER
0250 445A        967          Jmp     DclR2
                 968
                 969 ; forward:
                 970 ;        Set up for next phase bit output before entering timing loops
0252 1B          971 DclF2:   Inc     PhzR30            ;increment the phase addr
0253 FB          972          MOV     A,PhzR30          ;Get the phz data addr
0254 5258        973          JB2     DZroPh            ;CHK FOR COUNT BIT ROLLOVER
0256 445A        974          JMP     DNxtPh            ;skip adr index reset
0258 BB00        975 DZroPh:  MOV     PhzR30,#FStCRP    ;ZERO CR SM PHASE REGISTER
                 976 DNxtPh:                            ;set up for next phase shift
025A FB          977 DclR2:   MOV     A,PhzR30          ;get phase index address
025B E3          978          MovP3   A,@A              ;get phase from indexed address
                 979 ;        patch together the CR last and LF next phase bits
025C B820        980          Mov     TmpR00,#LastPh    ;load Last Phz psuedo reg to Temp Reg
025E 40          981          ORL     A,@TmpR00          ;patch together CR existing & new LF
025F 1663        982 TLoopD:  JTF     NxtPD2            ;JMP ON TIME OUT TO NEXT PH
0261 445F        983          JMP     TLoopD            ;LOOP UNTIL TIME OUT
0263 3C          984 NxtPD2:  MOVD    P4,A              ;OUTPUT BITS
0264 EC41        985          DJNZ    CntR40,StrtTD     ;Exit Test
                 986
                 987 ;        Set Storeage of next phase data in psuedo addr. This insures
                 988 ;        next phase is sequence correct for stpr mtr drive direction
0266 B821        989 SetRN:   Mov     TmpR00,#CPSAdr    ;get Phz Storeage Addr psuedo reg
0268 FB          990          MOV     A,PhzR30          ;get Phz data
0269 A0          991          Mov     @TmpR00,A         ;store CR Next phase index addr
026A B478        992 DMExit:  Call    DlyLng
026C B490        993          Call    DeSlSM
026E 83          994          RET
                 995
                 996 ; PG
                 997 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 998 ;       Stepper Motor Phase Shift Definitions
                 999 ;          All program procedures call this data.
                1000 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                1001
0300            1002          ORG     300H
                1003
                1004 ;        DEFINE PHASE ADDRESSES:
                1005 ;        THE PHASE DATA IS ENCODED TO THE ADDRESS CALLED DURING THE
                1006 ;        STPR MTR ENERGIZE SEQUENCE CORRESPONDING TO THE NEXT PHASE
                1007 ;        OF THE SEQUENCE REQUIRED.
                1008
                1009 ;        CARRAGE MOTOR ENCODING: FORWARD  -  LEFT-to-RIGHT
                1010 ;                                REVERSE  -  RIGHT-to-LEFT
                1011
```

230795-41

```
                        1012 ;       Reverse direction ENCODING is the same bytes accessed in
                        1013 ;         reverse direction
                        1014 ;
0300 01                 1015         DB      CRMFP1
0301 03                 1016         DB      CRMFP2
0302 02                 1017         DB      CRMFP3
0303 00                 1018         DB      CRMFP4
                        1019 ;
                        1020 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1021 ;
                        1022 ;       LF MOTOR PHASE ENCODE & DECODE: FORWARD (CLOCKWISE)
                        1023 ;       Forward direction ENCODING:
                        1024 ;       ---------------------------------------------
                        1025 ;
0308                    1026         ORG     308H
                        1027 ;
0308 04                 1028         DB      LFMFP1
0309 0C                 1029         DB      LFMFP2
030A 08                 1030         DB      LFMFP3
030B 00                 1031         DB      LFMFP4
                        1032 ;
                        1033 ;
                        1034 ; PG
                        1035 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1036 ;       Accel/Decel / Character Handling Test
                        1037 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1038 ;       TEST  >  Is CR Stpr Mtr At Speed ??
                        1039 ;         Yes - SetUp do Character Processing
                        1040 ;         No  - Calculate / Store the Acceleration Phase Shift Time (11)
                        1041 ; ---------------------------------------------------------------------------
                        1042 ;
030C FA                 1043 ADPTst: Mov     A,QStR2O        ;get the status byte
030D B211               1044         JB5     PHFSet          ;test if Stpr Mtr At Speed
                        1045 ;                               ;   jmp to Prnt Head Fire Setup
030F 4400               1046         Jmp     ADMmTS          ;else Call Accel/Decel Memory Time Store
                        1047 ;
                        1048 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1049 ;       Process Characters for Printing
                        1050 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1051 ;
                        1052 ;       Character dot matrix - normal char
                        1053 ;       d = Dot Column
                        1054 ;       b = Blank Column
                        1055 ;
                        1056 ;       b d d d d d
                        1057 ;       (Char Matrix)
                        1058 ;       0 0 0 0  b
                        1059 ;       0 0 0 1  d
                        1060 ;       0 0 1 0  d
                        1061 ;       0 0 1 1  d
                        1062 ;       0 1 0 0  d
                        1063 ;       0 1 0 1  d
                        1064 ; ---------------------------------------------------------------------------
                        1065 ;
0311 2668               1066 PHFSet: JNTO    Retrn           ;if R=0 not ready to print-exit
0313 326A               1067         JB1     NPRet           ;if Do Not Print stat bit set - EXIT
0315 D21B               1068         JB6     SinkSt          ;if bit previously set-skip setting it
0317 FA                 1069         Mov     A,QStR2O        ;get the status byte
0318 4340               1070         ORL     A,#SnkSet       ;set Prnt Ready Sink bit
031A AA                 1071         Mov     QStR2O,A        ;save the status byte
031B D5                 1072 SinkSt: SEL     RB1
031C FA                 1073         Mov     A,ChStR1        ;get char status register addr
031D D23A               1074         JB6     PageCk          ;test Char Init Done, 1 = Print Dot
                        1075 ;                               ;   0 = Get Char
                        1076 ;
                        1077 ; PG
                        1078 ; ---------------------------------------------------------------------------
                        1079 ;       Call for Individual character processing:  mid line test if CR/(LF)
                        1080 ; ---------------------------------------------------------------------------
                        1081 GetChr:
                        1082 ;       test for CR/(LF) if it is the test position in the line
031F F1                 1083 CRChCk: Mov     A,@CAdrR1       ;get character
0320 03F3               1084         ADD     A,#CRCp1        ;test for Carriage Return
0322 C626               1085         JZ      CrLnCk          ;   if CR go service it
0324 6437               1086         Jmp     AsciC1          ;if not CR Insert Space Char
0326 FA                 1087 CRLnCk: Mov     A,ChStR1        ;get char status register addr
0327 F22B               1088         JB7     HlfLn           ;test Chr Stat Byte Returned
                        1089 ;                               ;   if bit 7 = 1 then Print L-to-R
0329 6432               1090         Jmp     SpFill          ;if R-to-L print skip exit upon CR detect
                        1091 ; ---------------------------------------------------------------------------
```

230795-42

```
                        1092 ;       if L-to-R printing exit the line if less than 1/2 line printed
032B FD                 1093 H1fLn:  Mov     A,CCntR1          ;load char cnt reg w/char bufr size
032C 03D9               1094         ADD     A,#H1fCp1         ;add the 2's cpl of 1/2 chr buf size
032E F632               1095         JC      LnPad             ;if CB>1/2 full set CR/LF stat bit for pad
                        1096                                   ;If CB<1/2 set buffer full stat bit
0330 648A               1097         Jmp     MdLnEx            ;mid-line exit
                        1098 SpFill:
0332 97                 1099 LnPad:  Clr     C                 ;clear carry flag
0333 2320               1100         Mov     A,#Space          ;insert a space char
0335 6438               1101         Jmp     ChIsrt            ;char inserted jmp over get char
                        1102 ; ---------------------------------------------------------------------
0337 F1                 1103 AsciC1: Mov     A,@CAdrR1         ;get character
0338 7498               1104 ChIsrt: Call    GChar1            ;call the char lookup/trns table
                        1105 ; ---------------------------------------------------------------------
                        1106 ;       fetch the char dot column data
                        1107 PageCk:                           ;page test for balance of char
033A FA                 1108         Mov     A,ChStR1          ;get the status byte
033B B241               1109         JB5     FxJmp1            ;fix jmp over page boundries
033D F4EB               1110         Call    ChrPg2            ;Ascii char 50 - 7F Hex
033F 6443               1111         Jmp     MtxTst            ;jump to Matrix Test
0341 D4F0               1112 FxJmp1: Call    ChrPg1            ;Ascii char 20 - 4F Hex
                        1113                                   ;  fall thru to print matrix
                        1114                                   ;  and CB count tests
                        1115
                        1116 ; PG
                        1117 ; ---------------------------------------------------------------------
                        1118 ;       test the Char dot column print matrix count and Char buffer count
                        1119 ; ---------------------------------------------------------------------
0343 EB61               1120 MtxTst: DJNZ    CDtCR1,PrntDt     ;test for dot col or blank
                        1121                                   ;status byte in A upon entry here
0345 FA                 1122         Mov     A,ChStR1          ;get the status byte
0346 53BF               1123         ANL     A,#CIntND         ;set Char Init NotDone stat Flag
0348 AA                 1124         Mov     ChStR1,A          ;store the status byte
0349 ED58               1125         DJNZ    CCntR1,NotLCh     ;dec char cnt-jmp if Not Last Char
034B 53FD               1126         ANL     A,#NCBF1n         ;if 0 reset stat bit Not CB Full Line
034D 53FE               1127         ANL     A,#C1ICBR         ;reset CB Reg Init Flag - do Init
034F AA                 1128         Mov     ChStR1,A          ;save the status byte
                        1129
0350 C5                 1130         SEL     RB0
0351 FA                 1131         Mov     A,GStR20          ;get Gen Status register addr
0352 53FE               1132         ANL     A,#NotRdy         ;clear the ready bit
0354 AA                 1133         Mov     GStR20,A          ;store the General Status Byte
0355 D5                 1134         SEL     RB1
0356 6468               1135         Jmp     Retrn             ;EXit
                        1136
                        1137 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                        1138 ;         (see GChar1 ASCII char code translation procedure)
                        1139 ; ---------------------------------------------------------------------
                        1140 NotLCh:                           ;A contains LR/RL bit properly set
0358 FA                 1141         Mov     A,ChStR1          ;get char status register addr
0359 F25E               1142         JB7     StpCh2            ;test Chr Stat Byte Returned
                        1143                                   ;  if bit 7 = 1 then Print L-to-R
035B 19                 1144 StpCh1: Inc     CAdrR1            ;Increment char data memory addr.
035C 6468               1145         Jmp     Retrn
035E C9                 1146 StpCh2: Dec     CAdrR1            ;Decrement char data memory addr.
035F 6468               1147         Jmp     Retrn             ;  fall thru to Get Char
                        1148
                        1149
                        1150 ; ---------------------------------------------------------------------
                        1151 ;       Re-Entry Exit point for same char:
                        1152 ;         (before returning step the matrix)
                        1153 ; ---------------------------------------------------------------------
                        1154 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                        1155 ;         (see GChar1 ASCII char code translation procedure)
                        1156 ; ---------------------------------------------------------------------
                        1157
                        1158 PrntDt:
0361 FA                 1159 PrnDir: Mov     A,ChStR1          ;get char status byte
0362 F267               1160         JB7     StpCD2            ;test Chr Stat Byte Returned
                        1161                                   ;  if bit 7 = 1 then Print L-to-R
0364 CC                 1162 StpCD1: Dec     CDotR1            ;reverse step char dot col index
                        1163                                   ;  addr if R-to-L print
0365 6468               1164         Jmp     Retrn             ;skip over L-to-R print addr inc
0367 1C                 1165 StpCD2: INC     CDotR1            ;forward step char dot col index
                        1166                                   ;  addr if L-to-R print
                        1167                                   ;EXIT
                        1168
                        1169 ; PG
```

230795-43

```
                    1170 ; ---------------------------------------------------------------------
                    1171 ;       Character Print SetUp Exit Procedures
                    1172 ; ---------------------------------------------------------------------
                    1173 ;       Clean Standard Exit
                    1174 ; ---------------------------------------------------------------------
0368 C5             1175 Retrn:  SEL     RB0
0369 83             1176         Ret                        ;EXIT - return w/ Reg Bank 0 Reset
                    1177
                    1178 ;       Do Not Print exit:  set Stpr Mtr drive routine count loop
036A D5             1179 NPRet:  SEL     RB1
036B FA             1180         Mov     A,ChStR1           ;get the status byte
036C F27C           1181         JB7     SkpNPI             ;test print direction
                    1182 ; Reverse
036E C5             1183         SEL     RB0
036F FA             1184         Mov     A,GStR20           ;get the status byte
0370 53BF           1185         ANL     A,#ClrSnk          ;reset the print ready bit- skips PHFire call
0372 83             1186         Ret
                    1187 ; Forward
0373 D27C           1188         JB6     SkpNPI             ;test for first PHFSet entry reg init
                    1189 ;       Initialize register variables upon first entry
                    1190 ;         end of count clears char to print bit in status byte
0375 4340           1191         ORL     A,#ChIntD          ;set Char Reg Init Done stat bit
0377 AA             1192         Mov     ChStR1,A           ;save the status byte
0378 BB07           1193         Mov     TmpR10,#07H        ;load CR stpr mtr count during NoPrnt
037A 6488           1194         Jmp     NPExit
037C E888           1195 SkpNPI: DJNZ    TmpR10,NPExit
037E FA             1196         Mov     A,ChStR1           ;get the status byte
037F 53BF           1197         ANL     A,#CIntND          ;reset - char init not done
0381 AA             1198         Mov     ChStR1,A           ;save the status byte
0382 C5             1199         SEL     RB0
0383 FA             1200         Mov     A,GStR20           ;get Gen Status register addr
0384 53FE           1201         ANL     A,#NotRdy          ;clear the ready bit
0386 AA             1202         Mov     GStR20,A           ;store the General Status Byte
0387 83             1203 NSetEx: Ret
0388 C5             1204 NPExit: SEL     RB0
0389 83             1205         Ret
                    1206
                    1207 ;       Mid-Line Exit
                    1208 ; ---------------------------------------------------------------------
                    1209 ;       EXIT - if CR and not > 1/2 line done during L-to-R print
038A FA             1210 MdLnEx: Mov     A,ChStR1           ;get the status byte
038B 53FD           1211         ANL     A,#NCBF1n          ;if 0 reset stat bit Not CB Full Line
038D 53FE           1212         ANL     A,#C1ICBR          ;reset CB Reg Init Flag - do Init
038F AA             1213         Mov     ChStR1,A           ;save the status byte
0390 C5             1214         SEL     RB0
0391 FA             1215         Mov     A,GStR20           ;get the RB0 status byte
0392 4302           1216         ORL     A,#DoNotP          ;set the Do Not Print Flag(for RAccel)
0394 53BF           1217         ANL     A,#ClrSnk          ;reset the print ready bit-exit FAccel
0396 AA             1218         Mov     GStR20,A           ;save the status byte
0397 83             1219         Ret
                    1220
                    1221 ; PG
                    1222 ; ---------------------------------------------------------------
                    1223 ;       Character Dot Generator Math
                    1224 ;         Look-up Table Page Vectoring
                    1225 ;         Print Head Firing
                    1226 ; ---------------------------------------------------------------
                    1227
0398 AE             1228 GCHAR1: MOV     StrCR1,A                   ;STORE THE CHAR
                    1229
                    1230 ;  screen for printable char [char +(cpl 20 Hex + 1 = E0 Hex)]
0399 03E0           1231         ADD     A,#0E0H
039B F69F           1232         JC      PrntCh
039D 64C9           1233         jmp     CntlCh             ;jmp to control char lookup table
039F 97             1234 PrntCh: Clr     C                  ;clear carry flag
03A0 FE             1235         Mov     A,StrCR1           ;get the char again
                    1236
                    1237 ;  screen for char page [char +(cpl 50 Hex + 1 = B0 Hex)]
                    1238 ;    if carry char on page 2 else page 1
03A1 03B0           1239         ADD     A,#0B0H
03A3 F6AE           1240         JC      Page2
                    1241
                    1242 ;       Page 1 Character -- ASCII 20 Hex thru 4F Hex
                    1243 ;         Correct offset for lookup table page
                    1244 ;         ((char + E0 Hex)*5 = Page 1 index addr)
                    1245 ; ---------------------------------------------------------------
03A5 FA             1246 Page1:  Mov     A,ChStR1           ;get the status byte
03A6 4320           1247         OrL     A,#ChOnP1          ;set the page rentry flag bit
03A8 AA             1248         Mov     ChStR1,A           ;store the status byte
03A9 FE             1249         Mov     A,StrCR1           ;get the char agian
03AA 03E0           1250         ADD     A,#0E0H            ;set page 1 relative 00 offset
03AC 64B8           1251         Jmp     Multi5             ;jump to address math function
                    1252
```

230795-44

```
                    1253 ;       Page 2 Character -- ASCII 20 Hex thru 4F Hex
                    1254 ;         Correct offset for lookup table page two's complement
                    1255 ;         of ASCII chr code LookUp Table page base char of 50H plus
                    1256 ;         char * 5 ((char + B0 Hex)*5 = Page 2 index addr)
                    1257 ; ---------------------------------------------------------------
03AE 97             1258 Page2:  Clr     C                  ;clear carry flag
03AF FA             1259         Mov     A,ChStR1           ;get the status byte
03B0 53DF           1260         AnL     A,#ChOnP2          ;set the page rentry flag bit
03B2 AA             1261         Mov     ChStR1,A           ;store the status byte
03B3 FE             1262         Mov     A,StrCR1           ;get the char agian
03B4 03B0           1263         ADD     A,#0B0H            ;set page 2 relative 00 offset
03B6 64B8           1264         Jmp     Multi5             ;fall thru to address math function
                    1265
                    1266 ;       Compute character page offset dot pattern index address
03B8 AE             1267 MULTI5: Mov     StrCR1,A           ;store the zero offset char
03B9 E7             1268         RL      A                  ;MULTIPLY CHR BY 5 TO
03BA E7             1269         RL      A                  ;  FIND THE ADDRESS
03BB 6E             1270         ADD     A,StrCR1           ;ADD 1 TO COMPLETE 5X
03BC AC             1271         MOV     CDotR1,A           ;SAVE THE ADDRESS
                    1272
                    1273 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                    1274 ;         (see GChar1 ASCII char code translation procedure)
                    1275 ; ---------------------------------------------------------------
03BD FA             1276         Mov     A,ChStR1           ;get char status byte
03BE F2C4           1277         JB7     LRPrn              ;test Chr Stat Byte Returned
                    1278                                    ;  if bit 7 = 1 then Print L-toR
03C0 FC             1279         MOV     A,CDotR1           ;get the char index addr
03C1 0304           1280         ADD     A,#RLPShf          ;add char offset - start at end
                    1281                                    ;of char, print it R-to-L
03C3 AC             1282         MOV     CDotR1,A           ;SAVE THE ADDRESS
                    1283
                    1284 ;       Set the status byte for Character SetUp done
                    1285 ; ---------------------------------------------------------------
03C4 FA             1286 LRPrn:  Mov     A,ChStR1           ;get the status byte
03C5 4340           1287         ORL     A,#ChIntD          ;set 1st char col test bit = 0
03C7 AA             1288         Mov     ChStR1,A           ;store the status byte
03C8 83             1289         Ret                        ;return w/status byte in A
                    1290 ;       test for non printable characters goes here
03C9 83             1291 CntlCh: Ret
                    1292
                    1293 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    1294 ;       Print Head Fire
                    1295 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    1296
                    1297 ;       Entry point for print head solenoid firing
                    1298 ;         - test for status byte for dot/blank column position
03CA D5             1299 PHFire: SEL     RB1
03CB FB             1300         Mov     A,CDtCR1           ;set the chr dot column cnt
03CC 96D2           1301         JNZ     Fire               ;if char cnt not 0 - Fire Head Sol.
                    1302                                    ;if Chr Dot Cnt 0, reset the
03CE BB06           1303 SetCnt: Mov     CDtCR1,#NDtCCt     ;  char dot column count
03D0 64D8           1304         Jmp     Retrn1             ;skip PH Fire
03D2 2340           1305 Fire:   MOV     A,#PTrgLo          ;get the Prnt Head Trigger byte
03D4 3A             1306         OUTL    P2,A               ;FIRE PRINT HEAD
03D5 23C0           1307         MOV     A,#PTrgHi          ;get the Prnt Head Trigger byte
03D7 3A             1308         OUTL    P2,A               ;FIRE PRINT HEAD
03D8 C5             1309 Retrn1: SEL     RB0
03D9 83             1310         Ret                       ;EXIT - return w/ Reg Bank 0 Reset
                    1311
                    1312 ; PG
                    1313 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    1314 ;       PaperFeed Stpr Mtr Drive
                    1315 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    1316
0400                1317         ORG     400H
                    1318
                    1319 ;       Init psuedo register with LF inderect addr start - subsequent
                    1320 ;         exchanges of the psuedo register will yield correct value
0400 BC04           1321 InitLF: MOV     CntR40,#ILFCNT     ;INIT PHASE COUNT REG
0402 BB22           1322         Mov     TmpR00,#LPSAdr     ;get Phz Inderect Addr psuedo reg
0404 2308           1323         MOV     A,#StLFF           ;get LF starting addr
0406 A0             1324         Mov     @TmpR00,A          ;store LF phase index addr start
                    1325                                    ;  in psuedo register
0407 BE01           1326         Mov     LnCtR0,#LineCt     ;set line count reg for 1 ln
                    1327                                    ; enables exit following LF SM init
0409 841B           1328         Jmp     LfDrv1             ;jump over line/form feed amd variable
                    1329                                    ;  line spacing tests & setups
                    1330
                    1331 ;       LineFeed / FormFeed Drive
```

230795-45

```
                        1332 ;  ---------------------------------------------------------------------------
                        1333
                        1334 ;       load step count constant for standard line spacing
                        1335 ;  ---------------------------------------------------------------------------
                        1336 ;       test for various line/inch spacing would go here
                        1337 ;         (and removal of constant setup below)
0408 BC1B               1338         MOV     CntR40,#LPI8p8  ;init cnt reg for standard line feed
                        1339
                        1340 ;       LineFeed/FormFeed Test
040D FA                 1341 LfDriv: Mov     A,@StR20        ;get the status byte
040E 5214               1342         JB2     FmFd            ;if linefeed jmp to cnt load
0410 BE01               1343 LnCtLd: Mov     LnCtR0,#LineCt  ;set line count reg for 1 line
0412 841B               1344         Jmp     LfDrv1          ;jmp to Start of Drive
0414 FE                 1345 FmFd:   Mov     A,LnCtR0        ;get the line count
0415 37                 1346         Cpl     A               ;2's cpl Line Count
0416 0301               1347         Add     A,#01
0418 0342               1348         Add     A,#PgLnCt       ;Add 2's cpl for Paging
                        1349                                 ;  PgLnCt - LnCt = n Lines to move
                        1350                                 ;  PgLnCt+(cpl(LnCt) = n lines to move
041A AE                 1351         Mov     LnCtR0,A        ;set the line count for FF
                        1352
                        1353 ;       for stablization of unused stpr mtr during CR stpr mtr drive,
                        1354 ;         store the unused stpr mtr current phase bits
041B B821               1355 LFDrv1: Mov     TmpR00,#CPSAdr  ;get the CR phz storeage addr
041D F0                 1356         Mov     A,@TmpR00       ;get the byute stored there
041E E3                 1357         MovP3   A,@A            ;get the phz data byte
041F B820               1358         Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0421 A0                 1359         Mov     @TmpR00,A       ;store Last Phase bits - inderect
                        1360 ;       exchange/store the phase register index addresses
0422 B822               1361         Mov     TmpR00,#LPSAdr  ;get Phz Inderect Addr psuedo reg
0424 F0                 1362         Mov     A,@TmpR00       ;get LF last phase index addr
0425 AB                 1363         Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
0426 BD98               1364         MOV     TConR0,#LFTMR1  ;Load time constant Reg
                        1365
                        1366 ;       Select the Stpr Mtr
0428 2306               1367         MOV     A,#SLF          ;GET CR SM SELECT BITS
042A 3D                 1368         MOVD    P5,A            ;SELECT SM [SCR80]
                        1369
                        1370 ;  ---------------------------------------------------------------------------
                        1371 ;       LineFeed / FormFeed Drive Loop
                        1372 ;  ---------------------------------------------------------------------------
042B F8                 1373         MOV     A,PhzR30        ;get the phz reg indirect addr index
042C E3                 1374         MovP3   A,@A            ;do indirect get of phz bits
                        1375 ;       patch together the CR last and LF next phase bits
042D B820               1376         Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
042F 40                 1377         ORL     A,@TmpR00       ;patch together CR existing & new LF
                        1378 ;       start timer and step motor
0430 3C                 1379         MOVD    P4,A            ;OUTPUT BITS
                        1380
                        1381 StrtLF:
0431 FD                 1382 STRLFT: MOV     A,TConR0        ;get time constant from reg
0432 62                 1383         MOV     T,A             ;load the timer
0433 55                 1384         STRT    T               ;START TIMER
                        1385 ;       setup the next phase to output
0434 1B                 1386         INC     PhzR30          ;STEP PHASE DB ADDRESS
0435 FB                 1387         MOV     A,PhzR30        ;get the phase index address
0436 523A               1388         JB2     ZROPHL          ;test phase
0438 843C               1389         JMP     NXTPHL
043A BB08               1390 ZROPHL: MOV     PhzR30,#STLFF   ;re-init phase register
                        1391
043C F8                 1392 NXTPHL: MOV     A,PhzR30        ;get the phz reg indirect addr index
043D E3                 1393         MovP3   A,@A            ;do indirect get of phz bits
                        1394 ;       patch together the CR last and LF next phase bits
043E B820               1395         Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0440 40                 1396         ORL     A,@TmpR00       ;patch together CR existing & new LF
                        1397
0441 1645               1398 TLoopL: JTF     NXPHLF          ;jmp on time out to output nxt phz
0443 8441               1399         JMP     TLOOPL          ;loop until timer times out
                        1400
0445 3C                 1401 NXPHLF: MOVD    P4,A            ;step motor - OUTPUT BITS
0446 EC31               1402         DJNZ    CntR40,StrLFT   ;test for end of phase count for line
                        1403                                 ;prep for next line
                        1404
                        1405 ;       test for various line/inch spacing would go here
0448 BC1B               1406         MOV     CntR40,#LPI8p8  ;init cnt reg for standard line feed
044A EE31               1407         DJNZ    LnCtR0,StrtLF   ;test for end of line count
                        1408
044C FA                 1409         Mov     A,@StR20        ;Get the status byte
044D 53FB               1410         ANL     A,#LineFd       ;reset for line feed
044F AA                 1411         Mov     @StR20,A        ;save the status byte
                        1412
                        1413 ;       store the phase register index addresses
                        1414 ;       Set LineFeed Stpr Mtr Next Phase index address
0450 B822               1415 SetLRN: Mov     TmpR00,#LPSAdr  ;get Phz Storage Addr psuedo reg
```

230795-46

```
0452 FB          1416          Mov     A,PhzR30          ;get the phase index address
0453 A0          1417          Mov     @TmpR00,A         ;store LF Next phase index addr
0454 B478        1418          Call    DlyLng
0456 B490        1419          Call    DeSISM
                 1420
                 1421 ;        Check if Char Buffer contains full line (80 char or nChar & CR)
                 1422 ;          exit otherwise continue to read in characters
                 1423 ;         Mov     A,GStR20          ;get the stat byte
                 1424 ;         JB1     ByPas1            ;if Do Not Print Bit Set - EXIT
                 1425 ;         Call    CBFck
0458 83          1426 ByPas1:  Ret
                 1427
                 1428 ; PG
                 1429 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1430 ;        Minor Software Subroutines
                 1431 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1432
0500             1433          ORG     500H
                 1434
                 1435 ; ---------------------------------------------------------------
                 1436 ;        System initialization subroutines
                 1437 ; ---------------------------------------------------------------
                 1438 Defalt:
                 1439 ; ---------------------------------------------------------------
                 1440 ;        reset/set EOF status flag bit = 0
0500 D5          1441          SEL     RB1
0501 FA          1442          Mov     A,ChStR1          ;get the char status byte
0502 53F7        1443          ANL     A,#ClrEOF         ;clear the EOF flag bit
0504 AA          1444          Mov     ChStR1,A          ;store the char status byte
0505 B823        1445          Mov     TmpR10,#PTAscS    ;get the Ascii code tmp store addr
0507 B020        1446          Mov     @TmpR10,#Ascii    ;load the tmp stor reg w/ascii start
0509 C5          1447          SEL     RB0
                 1448 ; ---------------------------------------------------------------
                 1449 ;        reset/set Ok-to-Print status flag bit = 0
050A FA          1450          Mov     A,GStR20          ;get the status byte
050B 53FD        1451          ANL     A,#OkPrnt         ;reset print flag - Ok Print
050D AA          1452          Mov     GStR20,A          ;save the status byte
050E 83          1453          RET
                 1454 InitAl:
                 1455 AllOff:
                 1456 ; ---------------------------------------------------------------
                 1457 ;        CLEAR all outputs
050F C5          1458          SEL     RB0
0510 230F        1459          MOV     A,#0FH            ;FORCE PORT HI - R/ OF 555
0512 3E          1460          MOVD    P6,A
0513 23FF        1461          MOV     A,#0FFH           ;TURN ALL PRNT SOL's OFF
0515 39          1462          OUTL    P1,A
0516 23C0        1463          MOV     A,#PTRGHI         ;print head fire tirgger inactive
0518 3A          1464          OUTL    P2,A
0519 8A03        1465          ORL     P2,#03            ;set comm hdsk to ACK hi/Busy hi
051B BA00        1466          Mov     GStR20,#00H       ;clear the status registers
051D D5          1467          SEL     RB1
051E BA00        1468          Mov     ChStR1,#00H
0520 C5          1469          SEL     RB0
0521 83          1470          RET                       ;RETURN TO INIT ROUTINE
                 1471
                 1472 ; PG
                 1473 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1474 ;        Home Carriage / Print Head Assembly
                 1475 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1476
0522 FA          1477 CRHome:  Mov     A,GStR20          ;get the status byte
0523 4302        1478          ORL     A,#DoNotP         ;set the do not print flag
0525 AA          1479          Mov     GStR20,A          ;save the status byte
0526 362A        1480          JTO     RtoL              ;test for position of PH assembly
                 1481                                    ;  drive accordingly
0528 3402        1482          Call    FAccel            ;drive CR Stpr Mtr
052A 3422        1483 RtoL:    Call    RAccel            ;find the logical left home CR position
                 1484                                    ;delay a long time before continuing
052C B474        1485          Call    DlyVLg
052E 83          1486          Ret
                 1487
                 1488 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1489 ;        Clear Data Memory
                 1490 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1491
                 1492 ;        At PowerUp or Reset, following CR & LF Stpr Mtr Init, this
                 1493 ;        procedure clears data memory above RB0, Stack and RB1.
052F B87F        1494 ClrDM:   MOV     R0,#DMTop         ;GET BUFFER START LOCATION [HEX]
0531 B95D        1495          MOV     R1,#DMSIZE
0533 B000        1496 ClrDM1:  MOV     @R0,#00H          ;ZERO MEMORY LOCATION
```

230795-47

```
0535 C8          1497          DEC     R0
0536 E933        1498          DJNZ    R1,ClrDM1       ;dec buffer, loop if not zero[end]
0538 83          1499          RET                     ;RETURN TO INIT ROUTINE
                 1500
                 1501 ; PG
                 1502 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1503 ;       Character Print TEST
                 1504 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1505
                 1506 PrnTst:
                 1507 ;  TEST ---  load the char buffer with successive increments of
                 1508 ;             the ascii code start.  test for end of ascii
                 1509 ;             printable chars and reinit the char stream loaded.
                 1510
0539 B97F        1511 CTInt:   Mov     CAdrR1,#FCBfSt  ;load char reg w/char bufr strt
053B BD50        1512          Mov     CCntR1,#ChBfSz  ;load char cnt reg w/char bufr size
                 1513 ChTst:                          ;Test char buffer fill with ASCII Char Code
053D FF          1514          Mov     A,opnr71        ;get the ascii char
053E A1          1515          Mov     @CAdrR1,A       ;load data memory w/Char
053F C9          1516          DEC     CAdrR1          ;Decrement dat memory location
0540 1F          1517          INC     opnr71          ;Increment Ascii char number
0541 0382        1518          ADD     A,#PAsEnd       ;test for ascii code end
0543 9647        1519          JNZ     ChrTGo          ;if not end jmp over code restart
0545 BF20        1520          Mov     OpnR71,#Ascii
0547 ED3D        1521 ChrTGo:  DJNZ    CCntR1,ChTst    ;dec buffer, loop if not zero[end]
0549 C5          1522          SEL     RB0
054A 83          1523          RET                     ;ELSE RETURN TO INIT ROUTINE
                 1524
                 1525 ; PG
                 1526 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1527 ;       CR Stpr Mtr Power On Initialization and
                 1528 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1529 ;       This routine drives the CR or LF stpr mtr for four phase
                 1530 ;         shifts for initialization.
                 1531 INITCR:                          ;POWER ON INIT STPR MTR
054B BC04        1532          MOV     CntR40,#PhCnt1  ;load phase cnt reg for INIT
054D 2308        1533          MOV     A,#SCR80        ;GET CR SM SELECT BITS
054F 3D          1534          MOVD    P5,A            ;SELECT SM [SCR80]
0550 BDC0        1535          MOV     TConRO,#IntTm2  ;Load time constant Reg
0552 BB00        1536          MOV     PhzR30,#FStCRP  ;zero SM phase reg - forward
0554 FB          1537          MOV     A,PhzR30        ;get phase index register byte
0555 E3          1538          MovP3   A,@A            ;load indexed phase shift byte
0556 3C          1539          MOVD    P4,A            ;OUTPUT BITS
0557 FD          1540 STRTTR:  MOV     A,TConRO        ;GET TIMER CONSTANT
0558 62          1541          MOV     T,A
0559 55          1542          STRT    T               ;START TIMER
055A 1B          1543          INC     PhzR30          ;step phase index register
055B FB          1544          MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
055C 5260        1545          JB2     ZroRg2
055E A462        1546          JMP     NxtPhR
0560 BB00        1547 ZroRg2:  MOV     PhzR30,#FStCRP  ;zero SM phase reg - forward
                 1548 NxtPhR:
0562 FB          1549          MOV     A,PhzR30        ;get phase index register byte
0563 E3          1550          MovP3   A,@A            ;load indexed phase shift byte
0564 1669        1551 TLoopR:  JTF     NXPHR1          ;JMP ON TIME OUT TO NEXT PH
0566 A464        1552          JMP     TLoopR          ;LOOP UNTIL TIME OUT
0568 3C          1553          MOVD    P4,A            ;OUTPUT BITS
0569 EC57        1554 NXPHR1:  DJNZ    CntR40,STRTTR
                 1555 ; ---------------------------------------------------------------------
                 1556 ;       store the last phase register index addresses
056B B821        1557          Mov     TmpROO,#CPSAdr   ;get Phz Storage Addr psuedo reg
056D FB          1558          Mov     A,PhzR30        ;place last CR phase index addr in Phz Reg
056E A0          1559          Mov     @TmpROO,A       ;  store CR last phase index addr
056F B478        1560          Call    DlyLng
0571 B490        1561          Call    DeSlSM
0573 83          1562          RET
                 1563
                 1564 ; PG
                 1565 ; ---------------------------------------------------------------------
                 1566 ;       Time Delay Subroutines
                 1567 ; ---------------------------------------------------------------------
                 1568
                 1569 ;       Very Long
0574 B87F        1570 DlyVLg:  MOV     TmpROO,#7FH     ;LOAD DELAY COUNT IN REG.
0576 A47E        1571          Jmp     DlyST
                 1572
                 1573 ;       Long
0578 B880        1574 DlyLng:  MOV     TmpROO,#DlyCL   ;LOAD DELAY COUNT IN REG.
057A A47E        1575          Jmp     DlyST
                 1576
```

230795-48

```
                        1577 ;       Not So Long - Short
057C B830               1578 DlySht: MOV     TmpR00,#DlyCS  ;LOAD DELAY COUNT IN REG.
                        1579
                        1580 ;       Start Delay
057E 23CC               1581 DlyST:  MOV     A,#DlyTim      ;GET MAX TIMER DELAY
0580 62                 1582 NxtTLd: MOV     T,A            ;LOAD TIMER
0581 55                 1583         STRT    T              ;START TIMER
                        1584
0582 168D               1585 DlyLop: JTF     DlyTO          ;LOOP
                        1586
                        1587 ;       Char buffer fill during time loop:
0584 D5                 1588         SEL     RB1
0585 FA                 1589         Mov     A,ChStR1       ;get the character stat reg byte
0586 928A               1590         JB4     SkpCI          ;  test for normal char input
                        1591                                ;  or skip if char prnt test
0588 1469               1592         Call    IBFSrv         ;service the char buffer fill
058A C5                 1593 SkpCI:  SEL     RB0
058B A482               1594         JMP     DlyLOP
058D E880               1595 DlyTO:  DJNZ    TmpR00,NxtTLd  ;dec delay count & test for exit
058F 83                 1596         RET
                        1597 ; -----------------------------------------------------------------------
                        1598 ;       Stpr Mtr Deselect
                        1599 ; -----------------------------------------------------------------------
                        1600 ;       Stepper Motor DeSelect Routine
                        1601 DESLSM:                       ;DESELECT LF/CR SM
0590 230E               1602 SMEROR: MOV     A,#SMOFF       ;GET LF/CR SM DE-SELECT BITS
0592 3D                 1603         MOVD    P5,A           ;DE-SELECT CR SM
0593 83                 1604         RET
                        1605
                        1606 $INCLUDE(:F1:CHRTBL.OV1)
                       =1607
                       =1608 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                       =1609 ;       Character Dot Generator Look-up Table Page 1
                       =1610 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                       =1611
                       =1612
                       =1613 ;             Character Table Page 1, contains
                       =1614
                       =1615 ;       20H ----------------------------------------> 4FH
                       =1616
                       =1617 ;    " (sp)!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM "
                       =1618
                       =1619 ; -----------------------------------------------------------------------
                       =1620
0600                   =1621         ORG     600H
                       =1622
                       =1623 ; -----------------------------------------------------------------------
                       =1624 ;       Page 1 --  Character Dot Pattern Fetch
                       =1625 ;       <<<  actual assembled character table code not listed  >>>
                       =1626 ; -----------------------------------------------------------------------
                       =1627 $NoList
                       =1676 $List
                       =1677 ;       Listing below is for reference only, actual code is not listed
                       =1678 ;         at assembly time.
                       =1679
                       =1680 ; -----------------------------------------------------------------------
                       =1681 ; asc20:        DB      7FH, 7FH, 7FH, 7FH, 7FH         ;SPACE
                       =1682 ; asc21:        DB      7FH, 7FH, 20H, 7FH, 7FH         ;!
                       =1683 ; asc22:        DB      7FH, 7FH, 78H, 7FH, 78H         ;"
                       =1684 ; asc23:        DB      6BH, 00H, 6BH, 00H, 6BH         ;#
                       =1685 ; asc24:        DB      5BH, 55H, 00H, 55H, 6DH         ;$
                       =1686 ; asc25:        DB      5CH, 6CH, 77H, 1BH, 1DH         ;%
                       =1687 ; asc26:        DB      19H, 26H, 26H, 59H, 2FH         ;&
                       =1688 ; asc27:        DB      7FH, 7FH, 7CH, 7FH, 7FH         ;'
                       =1689 ; asc28:        DB      63H, 5DH, 3EH, 7FH, 7FH         ;(
                       =1690 ; asc29:        DB      7FH, 7FH, 3EH, 5DH, 63H         ;)
                       =1691 ; asc2A:        DB      5DH, 6BH, 00H, 6BH, 5DH         ;*
                       =1692 ; asc2B:        DB      77H, 77H, 41H, 77H, 77H         ;+
                       =1693 ; asc2C:        DB      7FH, 3FH, 4FH, 7FH, 7FH         ;,
                       =1694 ; asc2D:        DB      77H, 77H, 77H, 77H, 77H         ;-
                       =1695 ; asc2E:        DB      7FH, 1FH, 1FH, 7FH, 7FH         ;.
                       =1696 ; asc2F:        DB      5FH, 6FH, 77H, 7BH, 7DH         ;/
                       =1697 ; asc30:        DB      41H, 2EH, 36H, 3AH, 41H         ;0
                       =1698 ; asc31:        DB      7FH, 3DH, 00H, 3FH, 7FH         ;1
                       =1699 ; asc32:        DB      3DH, 1EH, 2EH, 36H, 39H         ;2
                       =1700 ; asc33:        DB      5DH, 3EH, 36H, 36H, 49H         ;3
                       =1701 ; asc34:        DB      67H, 6BH, 6DH, 00H, 6FH         ;4
                       =1702 ; asc35:        DB      58H, 3AH, 3AH, 3AH, 46H         ;5
                       =1703 ; asc36:        DB      43H, 35H, 36H, 36H, 4EH         ;6
                       =1704 ; asc37:        DB      7EH, 0EH, 76H, 7AH, 7CH         ;7
                       =1705 ; asc38:        DB      49H, 36H, 36H, 36H, 49H         ;8
```

230795-22

```
                        =1706 ; asc39:      DB      39H, 36H, 36H, 56H, 61H         ;9
                        =1707 ; asc3A:      DB      7FH, 7FH, 6BH, 7FH, 7FH         ;:
                        =1708 ; asc3B:      DB      7FH, 3FH, 4BH, 7FH, 7FH         ;;
                        =1709 ; asc3C:      DB      77H, 6BH, 5DH, 3EH, 7FH         ;<
                        =1710 ; asc3D:      DB      6BH, 6BH, 6BH, 6BH, 6BH         ;=
                        =1711 ; asc3E:      DB      7FH, 3EH, 5DH, 6BH, 77H         ;>
                        =1712 ; asc3F:      DB      79H, 7EH, 26H, 7AH, 7DH         ;?
                        =1713 ; asc40:      DB      41H, 3EH, 22H, 36H, 71H         ;@
                        =1714 ; asc41:      DB      03H, 6DH, 6EH, 6DH, 03H         ;A
                        =1715 ; asc42:      DB      00H, 36H, 36H, 36H, 49H         ;B
                        =1716 ; asc43:      DB      41H, 3EH, 3EH, 3EH, 5DH         ;C
                        =1717 ; asc44:      DB      00H, 3EH, 3EH, 5DH, 63H         ;D
                        =1718 ; asc45:      DB      00H, 36H, 36H, 36H, 36H         ;E
                        =1719 ; asc46:      DB      00H, 76H, 76H, 76H, 76H         ;F
                        =1720 ; asc47:      DB      41H, 3EH, 3EH, 2EH, 0DH         ;G
                        =1721 ; asc48:      DB      00H, 77H, 77H, 77H, 00H         ;H
                        =1722 ; asc49:      DB      7FH, 3EH, 00H, 3EH, 7FH         ;I
                        =1723 ; asc4A:      DB      5FH, 3FH, 3FH, 3FH, 40H         ;J
                        =1724 ; asc4B:      DB      00H, 77H, 6BH, 5DH, 3EH         ;K
                        =1725 ; asc4C:      DB      00H, 3FH, 3FH, 3FH, 3FH         ;L
                        =1726 ; asc4D:      DB      00H, 7DH, 73H, 7DH, 00H         ;M
                        =1727 ; asc4E:      DB      0aaH, 0dfH, 0efH, 0f7H, 0aaH    ;test
                        =1728 ; asc4F:      DB      55H, 0dfH, 0efH, 0f7H, 55H      ;test
                        =1729 ; asc4E:      DB      00H, 7BH, 77H, 6FH, 00H         ;N
                        =1730 ; asc4F:      DB      41H, 3EH, 3EH, 3EH, 41H         ;O
                        =1731 ; ------------------------------------------------------------------------------
                        =1732 ;         End Page 1  --  Character Dot Pattern Fetch
                        =1733
                        =1734 ; ------------------------------------------------------------------------------
                        =1735 ;         Character Dot Pattern Fetch
                        =1736 ; ------------------------------------------------------------------------------
                        =1737
06F0 FC                 =1738 ChrPg1:  MOV     A,CDotR1        ;get char index address offset
06F1 A3                 =1739          MOVP    A,@A            ;get column dot patern byte
                        =1740
                        =1741 ;        this bit fix necessary to not underline each character
                        =1742 ;          this saves fixing each bit in the look up table
                        =1743
06F2 4380               =1744          ORL     A,#80H          ;char bit fix
06F4 39                 =1745          OutL    P1,A            ;output the dot pattern
06F5 83                 =1746          RET                     ;exit with byte in acc
                        =1747
                        =1748 ; ------------------------------------------------------------------------------
                        =1749 ;         END Page 1  --  Character Dot Pattern Fetch
                        =1750 ; ------------------------------------------------------------------------------
                        =1751
                        =1752
                        =1753 ;         PAGE 2  -- Character Dot Generator Look-Up Table
                        =1754 ; ------------------------------------------------------------------------------
                        =1755
                        =1756 ;                 Character Table Page 2, contains
                        =1757
                        =1758 ;         50H ----------------------------------------> 7EH
                        =1759
                        =1760 ;    " NOPQRSTUVWXYZ[\]^_(?)abcdefghijklmnopqrstuvwxyz{|}~ "
                        =1761
                        =1762 ; ------------------------------------------------------------------------------
                        =1763
0700                    =1764          ORG     700H
                        =1765
                        =1766 ; ------------------------------------------------------------------------------
                        =1767 ;         Page 2  --  Character Dot Pattern Fetch
                        =1768 ;         <<<  Actual assembled character table code not listed  >>>
                        =1769 ; ------------------------------------------------------------------------------
                        =1770 $NoLIST
                        =1818 $List
                        =1819 ;         Listing below is for reference only, actual code is not listed
                        =1820 ;           at assembly time.
                        =1821
                        =1822 ; ------------------------------------------------------------------------------
                        =1823 ; asc50:          DB      00H, 76H, 76H, 76H, 79H         ;P
                        =1824 ; asc51:          DB      41H, 3EH, 2EH, 5EH, 21H         ;Q
                        =1825 ; asc52:          DB      00H, 76H, 66H, 56H, 39H         ;R
                        =1826 ; asc53:          DB      59H, 36H, 36H, 36H, 4DH         ;S
                        =1827 ; asc54:          DB      7EH, 7EH, 00H, 7EH, 7EH         ;T
                        =1828 ; asc55:          DB      40H, 3FH, 3FH, 3FH, 40H         ;U
                        =1829 ; asc56:          DB      60H, 5FH, 3FH, 5FH, 60H         ;V
                        =1830 ; asc57:          DB      00H, 5FH, 67H, 5FH, 00H         ;W
                        =1831 ; asc58:          DB      1CH, 6BH, 77H, 6BH, 1CH         ;X
                        =1832 ; asc59:          DB      7CH, 7BH, 07H, 7BH, 7CH         ;Y
```

230795-50

```
                =1833 ; asc5A:      DB      1EH, 2EH, 36H, 3AH, 3CH         ; Z
                =1834 ; asc5B:      DB      00H, 3EH, 3EH, 3EH, 7FH         ; [
                =1835 ; asc5C:      DB      7DH, 7BH, 77H, 6FH, 5FH         ; \
                =1836 ; asc5D:      DB      7FH, 3EH, 3EH, 3EH, 00H         ; ]
                =1837 ; asc5E:      DB      6FH, 77H, 7BH, 77H, 6FH         ; ^
                =1838 ; asc5F:      DB      3FH, 3FH, 3FH, 3FH, 3FH         ; _
                =1839 ; asc60:      DB      7DH, 7BH, 77H, 0FFH, 0FFH       ; \
                =1840 ; asc61:      DB      0DFH, 0ABH, 0ABH, 0ABH, 087H    ; a
                =1841 ; asc62:      DB      080H, 0B7H, 0B7H, 0B7H, 0CFH    ; b
                =1842 ; asc63:      DB      0C7H, 0BBH, 0BBH, 0BBH, 0BBH    ; c
                =1843 ; asc64:      DB      0CFH, 0B7H, 0B7H, 0B7H, 080H    ; d
                =1844 ; asc65:      DB      0C7H, 0ABH, 0ABH, 0ABH, 0B7H    ; e
                =1845 ; asc66:      DB      0F7H, 081H, 0F6H, 0FEH, 0FDH    ; f
                =1846 ; asc67:      DB      0F7H, 0ABH, 0ABH, 0ABH, 0C3H    ; g
                =1847 ; asc68:      DB      080H, 0F7H, 0FBH, 0FBH, 087H    ; h
                =1848 ; asc69:      DB      0FFH, 0BFH, 0BBH, 0BFH, 0FFH    ; i
                =1849 ; asc6A:      DB      0DFH, 0BFH, 0BBH, 0C2H, 0FFH    ; j
                =1850 ; asc6B:      DB      0FFH, 080H, 0EFH, 0D7H, 0BBH    ; k
                =1851 ; asc6C:      DB      0FFH, 0BEH, 080H, 0BFH, 0FFH    ; l
                =1852 ; asc6D:      DB      087H, 0FBH, 0E7H, 0FBH, 087H    ; m
                =1853 ; asc6E:      DB      083H, 0F7H, 0FBH, 0FBH, 087H    ; n
                =1854 ; asc6F:      DB      0C7H, 0BBH, 0BBH, 0BBH, 0C7H    ; o
                =1855 ; asc70:      DB      084H, 0EBH, 0EBH, 0EBH, 0F7H    ; p
                =1856 ; asc71:      DB      0F7H, 0EBH, 0EBH, 0EBH, 084H    ; q
                =1857 ; asc72:      DB      0FFH, 083H, 0F7H, 0FBH, 0FBH    ; r
                =1858 ; asc73:      DB      0B7H, 0ABH, 0ABH, 0ABH, 0DBH    ; s
                =1859 ; asc74:      DB      0FBH, 0C1H, 0BBH, 0DFH, 0FFH    ; t
                =1860 ; asc75:      DB      0C3H, 0BFH, 0BFH, 0BFH, 0C3H    ; u
                =1861 ; asc76:      DB      0E3H, 0DFH, 0BFH, 0DFH, 0E3H    ; v
                =1862 ; asc77:      DB      0C3H, 0BFH, 0CFH, 0BFH, 0C3H    ; w
                =1863 ; asc78:      DB      0BBH, 0C7H, 0EFH, 0C7H, 0BBH    ; x
                =1864 ; asc79:      DB      0FFH, 0B3H, 0AFH, 0AFH, 0C3H    ; y
                =1865 ; asc7A:      DB      0BBH, 09BH, 0ABH, 0B3H, 0BBH    ; z
                =1866 ; ASC7B:      DB      07FH, 077H, 049H, 03EH, 03EH    ; {
                =1867 ; ASC7C:      DB      0FFH, 0FFH, 088H, 0FFH, 0FFH    ; |
                =1868 ; ASC7D:      DB      03EH, 03EH, 009H, 077H, 07FH    ; }
                =1869 ; ASC7E:      DB      067H, 07BH, 067H, 05FH, 067H    ; ~
                =1870
                =1871 ; ------------------------------------------------------------------
                =1872 ;       Character Dot Pattern Fetch
                =1873 ; ------------------------------------------------------------------
                =1874
07EB FC         =1875 ChrPg2: MOV     A,CDotR1          ;get char index address offset
07EC A3         =1876         MOVP    A,@A              ;get column dot patern byte
                =1877
                =1878 ;       this bit fix necessary to not underline each character
                =1879 ;         this saves fixing each bit in the look up table
                =1880
07ED 4380       =1881         ORL     A,#80H            ;char bit fix
07EF 39         =1882         OutL    P1,A              ;output the dot pattern
07F0 83         =1883         RET                       ;exit with byte in acc
                 1884
                 1885
                 1886
                 1887 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1888 ;                 Program End
                 1889 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1890
                 1891         END

ASSEMBLY COMPLETE,   NO ERRORS
```

230795-51

APPENDIX B
SOFTWARE PRINTER ENHANCEMENTS

This section describes several software enhancements which could be implemented as additions to the software developed for this Application Note. Space is available for most of the items described. Approximately 5 bytes of Data Memory would be required to implement most of the features. Two bytes would be used for status flags, and two bytes for temporary data or count storage. It is possible to use less than five bytes, but this would require the duplicate use of some flags, or other Data Memory storage, which will significantly complicate the software coding and debug tasks.

Special Characters or Symbols

Dot matrix printing lends itself well to the creation of custom characters and symbols. There are two aspects to implementing special characters. First, a character look-up table, and second, additional software for decoding and processing the special characters or symbols. Special characters might be scientific notation, mathematical symbols, unique language characters, or block and line graphics characters.

The character look-up table could be an additional page of Program Memory dedicated to the special characters, or replace part, or all, of the existing look-up tables. If an additional look-up table is used, a third page test would be needed at the beginning of the Character Translation subroutine. There is fundamentally no difference between the processing of special characters and standard ASCII printable characters. If the characters require the same 5 x 7 dot matrix, the balance of the software would remain the same. If, however, the special characters require a different matrix, or the manipulation of the matrix, the software becomes more complex.

In general, the major software modification required to implement special characters is the size of the dot matrix printed or the dot matrix configuration used. In the case of scientific characters, it would often be necessary to shift the 5 x 7 matrix pattern within the available 9 x 9 matrix. Block or line graphics characters, on-the-other-hand, would require using the entire 9 x 9 print head matrix and printing during normally blank dot columns. This would require suspending the blank column blanking mechanism implemented in this Application Note. This would be the most complex aspect of implementing special characters. It would possibly change the number of required instructions, and thus the timing between PTS detection and print head solenoid trigger firing. This could cause the dot columns to be misaligned within a printed line and between lines.

In the case of a matrix change, two approaches are possible: dynamically changing the matrix, in line, as standard ASCII characters are being printed, or isolating the special characters to a separate processing flow where special characters are handled as a unique and complete line of characters only. A discussion of in line matrix changes for special characters is beyond the scope of this Appendix. It is sufficient to say that the changes would require the conditions setting the EOLN flag, character count, and dot column count software be modified during character processing and printing.

Lower Case Descenders

The general principle of implementing lower case descenders is to shift the 5 x 7 character dot matrix within the available 9 x 9 print head solenoid matrix. Implementing lower case descenders requires two software modifications and the creation of status flag for the purpose. First, the detection of characters needing descenders and setting a dedicated status flag during the character code to dot pattern translation subroutine. Second, the character dot column data output to the print head solenoids must be shifted for each dot column of the character. At the end of the character, the flag would be reset.

Inline Control Codes

Inline control codes are two to three character sequences, which indicate special hardware conditions or software flow control and branching. The first character indicates that the control code sequence is beginning and is typically an ASCII Escape Character (ESC), 1BH. Termination of the inline code sequence would be indicated by a default number of code sequence characters. This would decrease the buffer size available for characters. Full 80 character line buffering would require loading the Character Buffer with a received character as a character is removed from it and processed.

The Inline Control Code test would be performed in two places: in the Character Buffer Fill subroutine and in the Character Processing (translation) subroutine. The test would be performed in the same manner that a Carriage Return (CR) character code test is implemented. Examples are horizontal tabs and expanded or con-

densed character fonts. In the case of horizontal tabs, 20H (Space Character) would have to be placed in the Character Buffer for inline processing during character processing and printing. Unless fixed position tabs are used, a minimum of a nibble of Data Memory would be required to maintain a "spaces-to-tab" count. Fixed tab positions could be set via another inline control code, by default of the print software, or through the use of external hardware switch settings. The control code method of setting the tab positions is the most desirable, but the most complex to implement.

Different Character Formats

Condensed and enlarged characters are variations in either the number of dots and/or the space used to print them. Thus, each character is a variation of the stepper motor and/or print head solenoid trigger timings. It is possible to print each in bold face by printing each dot twice per dot column position. This would require little software modification, but would require a status flag. Again, care must be used to ensure that the delay in retriggering the solenoids is precisely the same for each type of event. Without this precise timing the dot column alignment will not be accurate. The software modifications needed to implement enlarged or condensed characters is essentially the same. The carriage and print head solenoid firing software flow is the same, but the timing for each changes. For condensed characters, the step Time Constant is doubled to approximately 4.08 ms, and the solenoids are fired four times within each step time. The step rate actually becomes a multiple of the solenoid firing time, and a counter incrementing once for each solenoid firing would be needed. At the count of four, the carriage stepper motor is stepped and the counter reset.

In the case of condensed characters, PTS does not play the same roll as in standard or enlarged character printing. PTS is not used to indicate the optimum print head solenoid firing time. Solenoid firing is purely a time function for condensed characters. PTS would only be used for Failsafe protection.

Enlarged characters would require the solenoids be fired twice per dot column data, in two sequential dot columns, at the same rate as standard characters. The character dot column data and dot column count would not be incremented at each output but at every other output. A flag could be used for this purpose.

When printing either condensed or enlarged characters, the maximum character count would have to compensate for the increased or decreased characters per line count. When printing enlarged characters, the maximum characters per line would be 40.

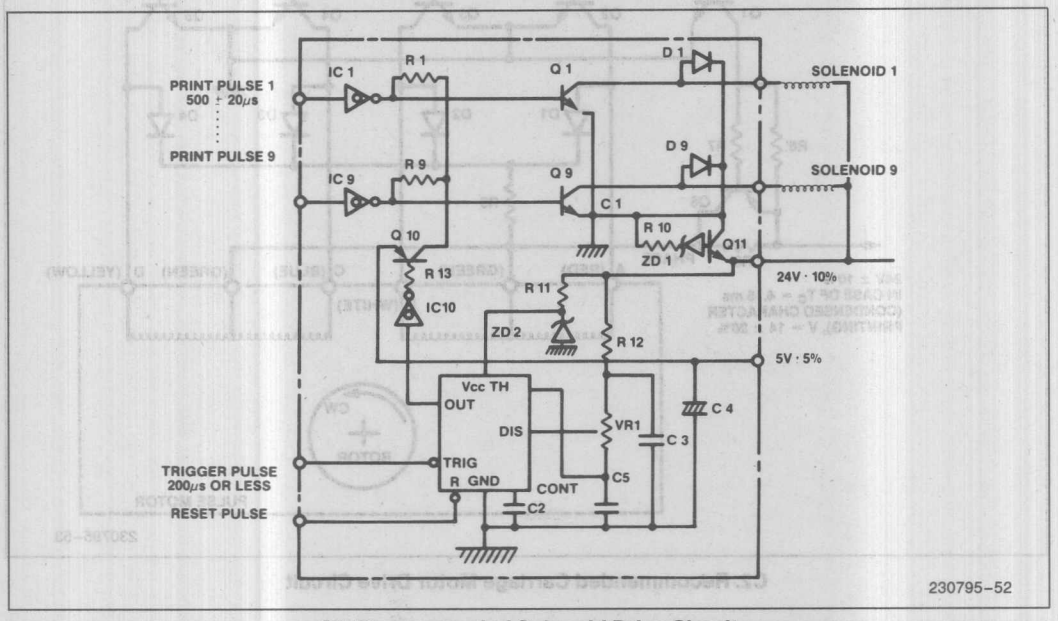
The Character Buffer could hold two complete lines of characters. But, condensed characters presents a quite different situation. The avalable character per line increases to 132, well beyond the 80 character Character Buffer size. The solution is to re-initialize the Character Buffer Size Count register count during condensed character processing. This will effectively inhibit the carriage stepper motor drive EOLN detection.

Two status flags would be required; one for standard or enlarged characters, and the second for condensed characters. A third status flag would be required to implement bold face printing. Activating one of the alternate character fonts could be either through the use of external status switches or through inline control code sequences, as detailed above. Note, that if the alternate character fonts are implemented in such a way that format changing is to occur dynamically during any single line being printed, the same control code problems described above also apply. In addition, the effect on the timing and dot column alignment must also be investigated.

Variable Line Spacing

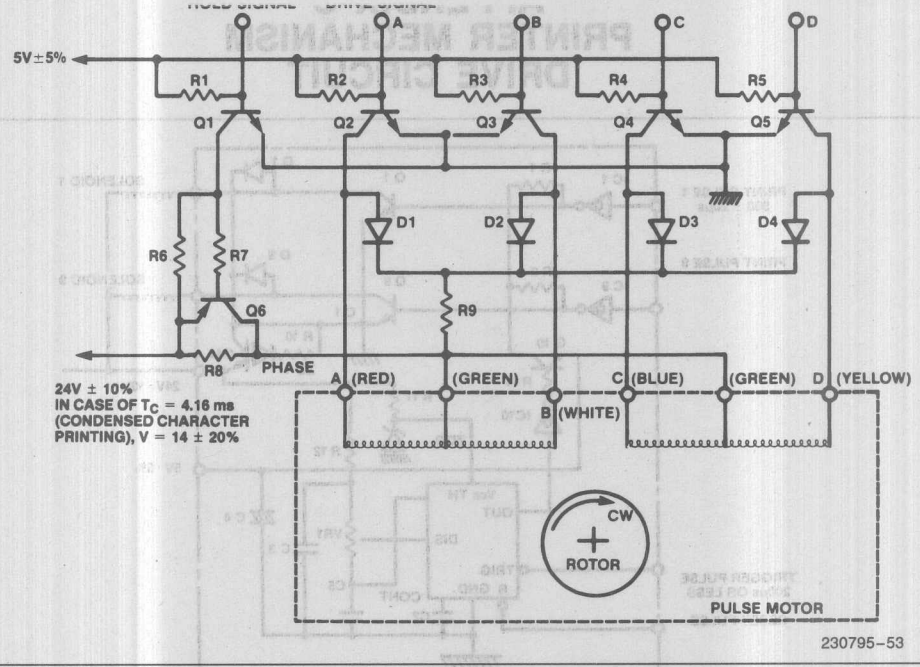
Variable line spacing is another feature which could be implemented either through the use of external status switches or inline control codes. The line spacing is a function of the number of steps the stepper motor rotates for a given line. Figure 15, Paper Feed Stepper Motor Predetermined Time Constants, in the Background section, lists the Time Constants required for three different line spacings; 6, 8, and 10 lines per inch. At the beginning of the Paper Feed Stepper Motor Drive subroutine, the default line step count is loaded. The software required is a conditional load for the line spacing, indicated by a status flag set in the External Status Switch Check subroutine or the Character Buffer Fill subroutine. Implementing the three different line spacings would require two additional status flags.

APPENDIX C
PRINTER MECHANISM DRIVE CIRCUIT



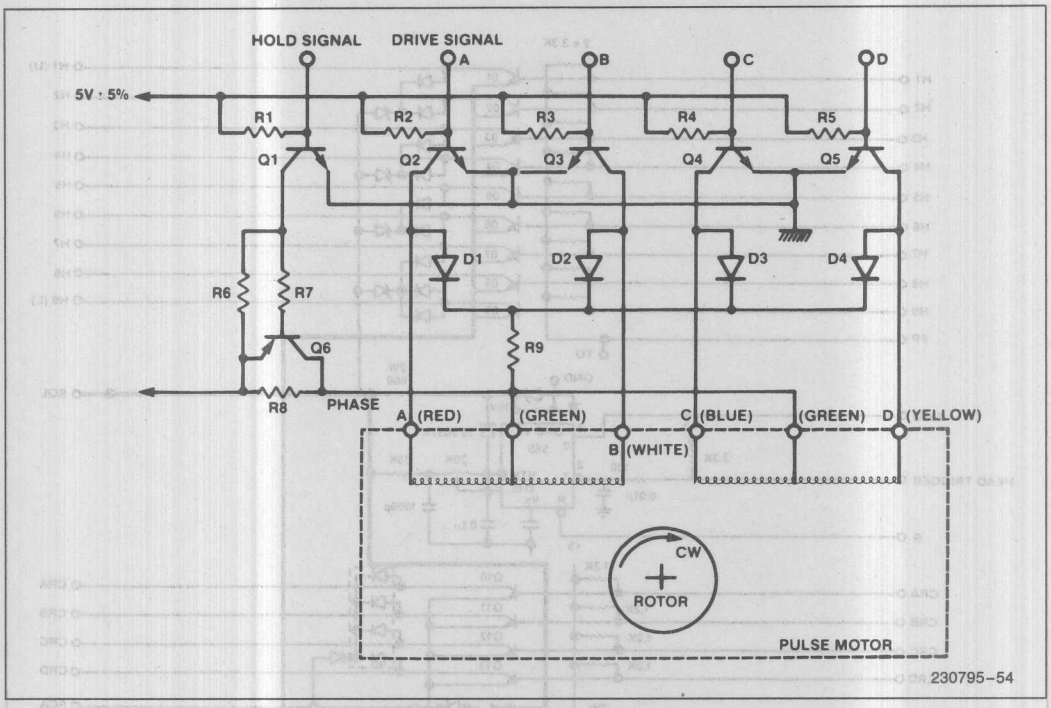
C1. Recommended Solenoid Drive Circuit

Parts No.		Type	Maker
IC1 ~ IC10		SN7406	TI
IC11		μA555	Fairchild
D1 ~ D9	Diode	S5277B	Toshiba
Q1 ~ 9	Transistor	2SD986	NEC
Q10	Transistor	2SA1015	Toshiba
Q11	Transistor	2SD633	Toshiba
R1 ~ R9	Resistor	1.2 kΩ ¼	
R10	Resistor	22Ω ¼	
R11	Resistor	580Ω 2	
R12	Resistor	15 kΩ ¼ Carbon fil =	
R13	Resistor	1.2 kΩ ¼	
VR1	Variable Resistor	20 kΩ ¼	
C1	Capacitor	1 μF 100V	
C2	Capacitor	0.01 μF	
C3	Capacitor	0.001 μF	
C4	Capacitor	10 μF 16V	
C5	Capacitor	0.1 μF fil =	
ZD1	Zenor Diode	HZ24	Hitachi
ZD2	Zenor Diode	HZ5C1	Hitachi



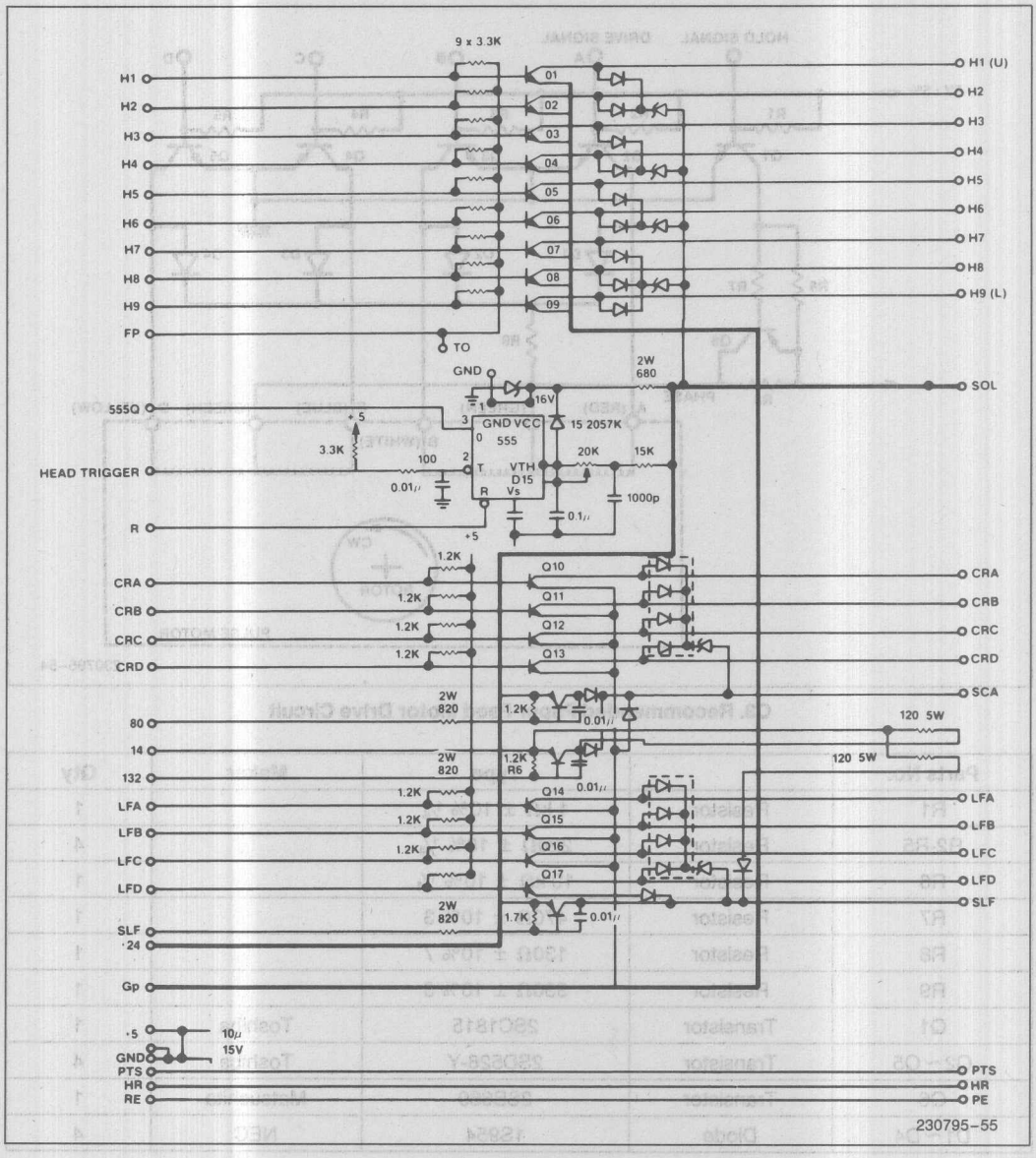
C2. Recommended Carriage Motor Drive Circuit

Parts No.		Type	Maker	Qty
R1	Resistor	1 kΩ ± 10% ¼		1
R2-R5	Resistor	220Ω ± 10% ¼		4
R6	Resistor	10 kΩ ± 10% ¼		1
R7	Resistor	470Ω ± 10% 3		1
R8	Resistor	130Ω ± 10% 7		1
R9	Resistor	330Ω ± 10% 3		1
Q1	Transistor	2SC1815	Toshiba	1
Q2 ~ Q5	Transistor	2SD526-Y	Toshiba	4
Q6	Transistor	2SB669	Matsushita	1
D1 ~ D4	Diode	1S954	NEC	4



C3. Recommended Paper Feed Motor Drive Circuit

Parts No.		Type	Maker	Qty
R1	Resistor	1 kΩ ± 10% ¼		1
R2-R5	Resistor	220Ω ± 10% ¼		4
R6	Resistor	10 kΩ ± 10% ¼		1
R7	Resistor	470Ω ± 10% 3		1
R8	Resistor	130Ω ± 10% 7		1
R9	Resistor	330Ω ± 10% 3		1
Q1	Transistor	2SC1815	Toshiba	1
Q2 ~ Q5	Transistor	2SD526-Y	Toshiba	4
Q6	Transistor	2SB669	Matsushita	1
D1 ~ D4	Diode	1S954	NEC	4


9 x 3.3K
H1
H2
H3
H4
H5
H6
H7
H8
H9
FP
TO
01
02
03
04
05
06
07
08
09
H1 (U)
H2
H3
H4
H5
H6
H7
H8
H9 (L)
GND
2W
680
SOL
555Q
HEAD TRIGGER
R
3.3K
100
0.01µ
GND VCC
555
VTH
D15
Vs
16V
15 2057K
20K
15K
1000p
0.1µ
+5
CRA
CRB
CRC
CRD
1.2K
Q10
Q11
Q12
Q13
2W
820
80
14
132
LFA
LFB
LFC
LFD
SLF
24
Gp
R6
Q14
Q15
Q16
Q17
1.7K
120 5W
10µ
15V
PTS
HR
RE
PE
230795-55

C4

APPLICATION NOTE

AP-90

November 1986

An 8741AH/8041A Digital Cassette Controller

JOHN BEASTON, JIM KAHN
PERIPHERAL APPLICATIONS

Order Number: 231314-002

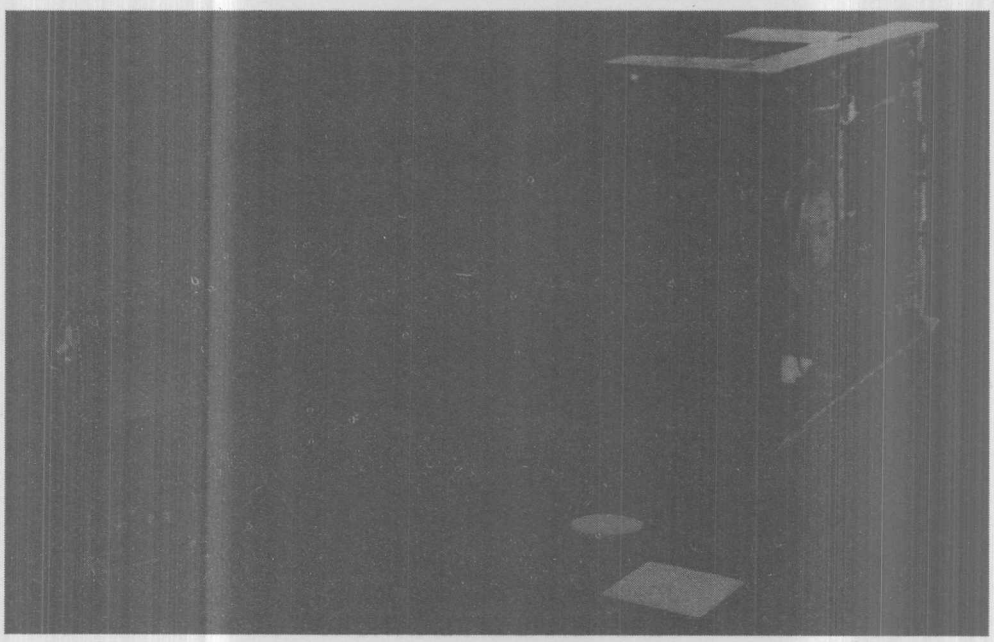
INTRODUCTION

The microcomputer system designer requiring a low-cost, non-volatile storage medium has a difficult choice. His options have been either relatively expensive, as with floppy discs and bubble memories, or non-transportable, like battery backed-up RAMs. The full-size digital cassette option was open but many times it was too expensive for the application. Filling this void of low-cost storage is the recently developed digital mini-cassette. These mini-cassettes are similar to, but not compatible with, dictation cassettes. The mini-cassette transports are inexpensive (well under $100 in quantity), small (less than 25 cu. in.), low-power (one watt), and their storage capacity is a respectable 200K bytes of unformatted data on a 100-foot tape. These characteristics make the mini-cassette perfect for applications ranging from remote datalogging to program storage for hobbyist systems.

The only problem associated with mini-cassette drives is controlling them. While these drives are relatively easy to interface to a microcomputer system, via a parallel I/O port, they can quickly overburden a CPU if other concurrent or critical real-time I/O is required. The cleanest and probably the least expensive solution in terms of development cost is to use a dedicated single-chip controller. However, a quick search through the literature turns up no controllers compatible with these new transports. What to do? Enter the UPI-41AH family of Universal Peripheral Interfaces.

The UPI-41AH family is a group of two user-programmable slave microcomputers plus a companion I/O expander. The 8741AH is the "flag-chip" of the line. It is a complete microcomputer with 1024 bytes of EPROM program memory, 64 bytes of RAM data memory, 16 individually programmable I/O lines, an 8-bit event counter and timer, and a complete slave peripheral interface with two interrupts and Direct Memory Access (DMA) control. The 8041A is the masked ROM, pin compatible version of the 8741AH. Figure 2 shows a block diagram common to both parts. The 8243 I/O port expander completes the family. Each 8243 provides 16 programmable I/O lines.

Using the UPI concept, the designer can develop a custom peripheral control processor for his particular I/O problem. The designer simply develops his peripheral control algorithm using the UPI-41AH assembly language and programs the EPROM of the 8741AH. Voila!



231314-1

Figure 1. Comparison of Mini-Cassette and Floppy Disk Transports and Media

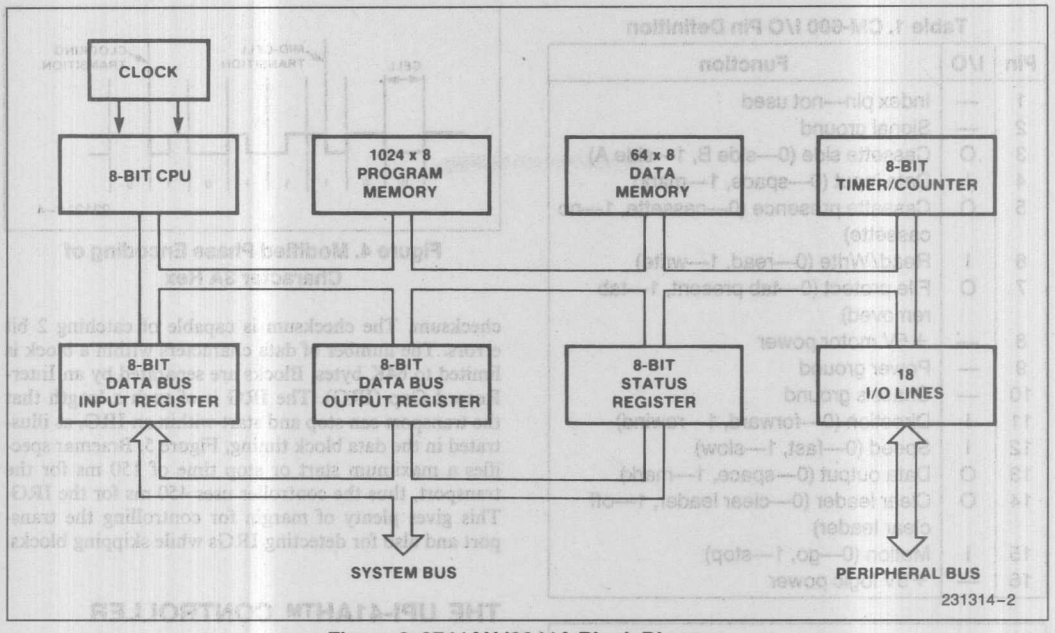


Figure 2. 8741AH/8041A Block Diagram

He has a single-chip dedicated controller. Testing may be accomplished using either an ICE-41A or the Single-Step mode of the 8741AH. UPI-41AH peripheral interfaces are being used to control printers, keyboards, displays, custom serial interfaces, and data encryption units. Of course, the UPI family is perfect for developing a dedicated controller for digital mini-cassette transports. To illustrate this application for the UPI family let's consider the job of controlling the Braemar CM-600 Mini-Dek*.

THE CM-600 MINI-DEK*

The Braemar CM-600 is representative of digital mini-cassette transports. It is a single-head, single-motor transport which operates entirely from a single 5V power supply. Its power requirements, including the motor, are 200 mA for read or write and 700 mA for rewind. Tape speeds are 3 inches per second (IPS) during read or write, 5 IPS fast forward, and 15 IPS rewind. With these speeds and a maximum recording density of 800 bits per inch (BPI), the maximum data rate is 2400 bits per second (BAUD). The data capacity using both sides of a 100-foot tape is 200K bytes. On top of this, the transport occupies only 22.5 cubic inches (3″ x3″ x2.5″).

*MiniDek is a registered trademark of Braemar.

All I/O for the CM-600 is TTL-compatible and can be divided into three groups: motor control, data control, and cassette status. The motor group controls are GO/STOP, FAST/SLOW, and FORWARD/REVERSE. The data controls are READ/WRITE, DATA IN, and DATA OUT. The remaining group of outputs give the transport's status: CLEAR LEADER, CASSETTE PRESENCE, FILE PROTECT, and SIDE SENSOR. These signals, shown schematically in Figure 3 and Table 1, give the pin definition of the CM-600 16-pin I/O connector.

RECORDING FORMAT

The CM-600 does not provide either encoding or decoding of the recorded data; that task is left for the peripheral interface. A multitude of encoding techniques from which the user may choose are available. In this single-chip dedicated controller application, a "self-clocking" phase encoding scheme similar to that used in floppy discs was chosen. This scheme specifies that a logic "0" is a bit cell with no transition; a cell with a transition is a logic "1."

Table 1. CM-600 I/O Pin Definition

Pin	I/O	Function
1	—	Index pin—not used
2	—	Signal ground
3	O	Cassette side (0—side B, 1—side A)
4	I	Data input (0—space, 1—mark)
5	O	Cassette presence (0—cassette, 1—no cassette)
6	I	Read/Write (0—read, 1—write)
7	O	File protect (0—tab present, 1—tab removed)
8	—	+5V motor power
9	—	Power ground
10	—	Chassis ground
11	I	Direction (0—forward, 1—rewind)
12	I	Speed (0—fast, 1—slow)
13	O	Data output (0—space, 1—mark)
14	O	Clear leader (0—clear leader, 1—off clear leader)
15	I	Motion (0—go, 1—stop)
16	—	+5V logic power

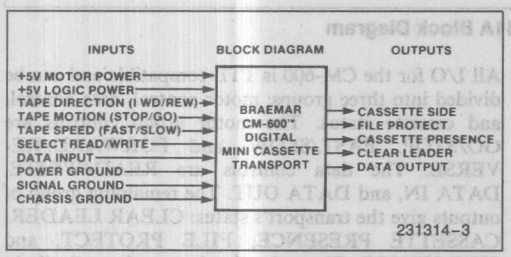


Figure 3. Braemar CM-600 Block Diagram

Figure 4 illustrates the encoding of the character 3AH assuming the previous data ended with the data line high. (The least significant bit is sent first.) Notice that there is always a "clocking" transition at the beginning of each cell. Decoding is simply a matter of triggering on this "clocking" transition, waiting ¾ of a bit cell time, and determining whether a mid-cell transition has occurred. Cells with no mid-cell transitions are data 0's; cells with transitions are data 1's. This encoding technique has all the benefits of Manchester encoding with the added advantage that the encoded data may be "decoded by eyeball:" long cells are always 0's, short cells are always 1's.

Besides the encoding scheme, the data format is also up to the user. This controller uses a variable byte length, checksum protected block format. Every block starts and ends with a SYNC character (AAH), and the character immediately preceding the last SYNC is the

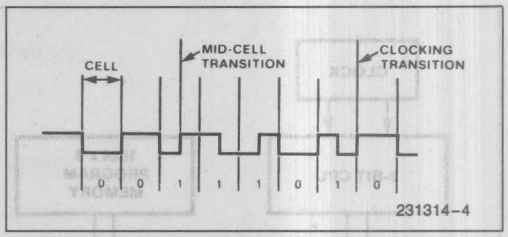


Figure 4. Modified Phase Encoding of Character 3A Hex

checksum. The checksum is capable of catching 2 bit errors. The number of data characters within a block is limited to 64K bytes. Blocks are separated by an Inter-Record Gap (IRG). The IRG is of such a length that the transport can stop and start within an IRG, as illustrated in the data block timing, Figure 5. Braemar specifies a maximum start or stop time of 150 ms for the transport, thus the controller uses 450 ms for the IRG. This gives plenty of margin for controlling the transport and also for detecting IRGs while skipping blocks.

THE UPI-41AH™ CONTROLLER

The goal of the UPI software design for this application was to make the UPI-41AH microcomputer into an intelligent cassette control processor. The host processor (8086, 8088, 8085A, etc.) simply issues a high-level command such as READ-a-block or WRITE-a-block. The 8741AH accepts the command, performs the requested operation, and returns to the host system a result code telling the outcome of the operation, eg. Good-Completion, Sync Error, etc. Table 2 shows the command and result code repertoire. The 8741AH completely manages all the data transfers for reading and writing.

As an example, consider the WRITE-a-block command. When this command is issued, the UPI-41AH expects a 16-bit number from the host telling how many data bytes to write (up to 64K bytes per block). Once this number is supplied in the form of two bytes, the host is free to perform other tasks; a bit in the UPI's STATUS register or an interrupt output will notify the host when a data transfer is required. The 8741AH then checks the transport's status to be sure that a cassette is present and not file protected. If either is false, a result code is returned to the host; otherwise the transport is started. After the peripheral controller checks to make sure that the tape is off the clear leader and past the hole in the tape, it writes a 450 ms IRG, a SYNC character, the block of data, the checksum, and the final SYNC character. (The tape has a clear leader at both ends and a small hole 6 inches from the end of each leader.) The data transfers from the host to the

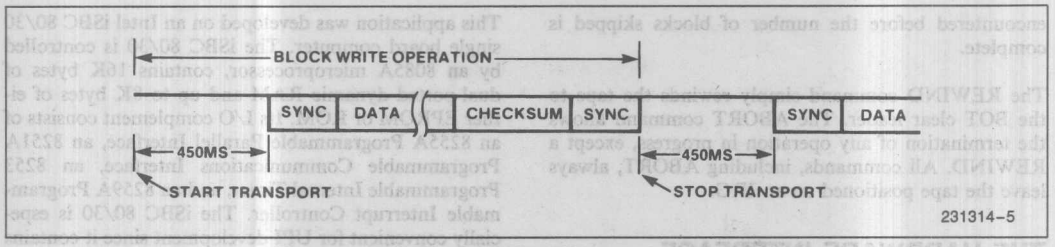


Figure 5. IRG/Block Timing Diagram (not to scale)

Table 2. Controller Command/Result Code Set

Command	Result
Read (01H)	Good-Completion (00H) Buffer Overrun Error (41H) Bad Synch1 Error (42H) Bad Synch2 Error (43H) Checksum Error (44H) Command Error (45H) End of Tape Error (46H)
Rewind (04H)	Good-Completion (00H)
Skip (03H)	Good-Completion (00H) End of Tape Error (47H) Beginning of Tape Error (48H)
Write (02H)	Good-Completion (00H) Buffer Underrun Error (81H) Command Error (82H) End of Tape Error (83H)

UPI-41AH slave microcomputer are double buffered. The controller requests only the desired number of data bytes by keeping track of the count internally.

If nothing unusual happened, such as finding clear leader while writing, it returns a Good-Completion result code to the host. If clear leader was encountered, the transport is stopped immediately and an End-of-Tape result code is returned to the host. Another possible error would be if the host is late in supplying data. If this occurs, the controller writes an IRG, stops the drive, and returns the appropriate Data-Underrun result code.

The READ-a-block command also provides error checking. Once this command is issued by the host, the controller checks for cassette presence. If present, it starts the transport. The data output from the transport is then examined and decoded continuously. If the first character is not a SYNC, that's an error and the controller returns a Bad-First-SYNC result code (42H) after advancing to the next IRG. If the SYNC is good, the succeeding characters are read into an on-chip 30 character circular buffer. This continues until an IRG is encountered. When this occurs, the transport is stopped. The controller then tests that it is the last character. If it is a SYNC, the controller then compares the accumulated internal checksum to the block's checksum, the second to the last character of the block. If they match, a Good-Completion result code (00H) is returned to the host. If either test is bad, the appropriate error result code is returned. The READ command also checks for the End-of-Tape (EOT) clear leader and returns the appropriate error result code if it is found before the read operation is complete.

The 30 character circular buffer allows the host up to 30 character times of response time before the host must collect the data. All data transfers take place thru the UPI-41AH Data Bus Buffer Output register (DBBOUT). The controller continually monitors the status of this register and moves characters from the circular buffer to the register whenever it is empty.

The SKIP-n-blocks command allows the host to skip the transport forward or backward up to 127 blocks. Once the command is issued, the controller expects one data byte specifying the number of blocks to skip. The most significant bit of this byte selects the direction of the skip (0 = forward, 1 = reverse). SKIP is a dual-speed operation in the forward direction. If the number of blocks to skip is greater than 8, the controller uses fast-forward (5 IPS until it is within 8 blocks of the desired location. Once within 8 blocks, the controller switches to the normal read speed (3 IPS) to allow accurate placement of the tape. The reverse skip uses only the rewind speed (15 IPS). Like the READ and WRITE commands, SKIP also checks for EOT and beginning-of-tape (BOT) depending upon the tape's direction. An error result code is returned if either is

encountered before the number of blocks skipped is complete.

The REWIND command simply rewinds the tape to the BOT clear leader. The ABORT command allows the termination of any operation in progress, except a REWIND. All commands, including ABORT, always leave the tape positioned on an IRG.

THE HARDWARE INTERFACE

There's hardly any hardware design effort required for the controller and transport interface in Figure 6. Since the CM-600 is TTL compatible, it connects directly to the I/O ports of the UPI controller. If the two are separated (i.e. on different PC cards), it is recommended that TTL buffers be provided. The only external circuitry needed is an LED driver for the DRIVE ACTIVE status indicator.

The 8741AH-to-host interface is equally straightforward. It has a standard asynchronous peripheral interface: 8 data lines (D_0–D_7), read (RD), write (WR), register select (A0), and chip select (CS). Thus it connects directly to an 8086, 8088, 8085A, 8080, or 8048 bus structure. Two interrupt outputs are provided for data transfer requests if the particular system is interrupt-driven. DMA transfer capability is also available. The clock input can be driven from a crystal directly or with the system clock (6 MHz max). The UPI-41AH clock may be asynchronous with respect to other clocks within the system.

This application was developed on an Intel iSBC 80/30 single board computer. The iSBC 80/30 is controlled by an 8085A microprocessor, contains 16K bytes of dual-ported dynamic RAM and up to 8K bytes of either EPROM or ROM. Its I/O complement consists of an 8255A Programmable Parallel Interface, an 8251A Programmable Communications Interface, an 8253 Programmable Interval Timer, and an 8259A Programmable Interrupt Controller. The iSBC 80/30 is especially convenient for UPI development since it contains an uncommitted socket dedicated to either an 8041A or 8741AH, complete with buffering for its I/O ports. The iSBC 80/30 to 8741AH interface is reflected in Figure 8. (Optionally, an iSBC 569 Digital Controller board could be used. The iSBC 569 board contains three uncommitted UPI sockets with an interface similar to that in Figure 8.)

Looking at the host-to-controller interface, the host sees the 8741AH as three registers in the host's I/O address space: the data register, the command register, and the status register. The decoding of these registers is shown in Figure 7. All data and commands for the controller are written into the Data Bus Buffer Input register (DBBIN). The state of the register select input, A0, determines whether a command or data is written. (Writes with A0 set to 1 are commands by convention.) All data and results from the controller are read by the host from the Data Bus Buffer Output register (DBBOUT).

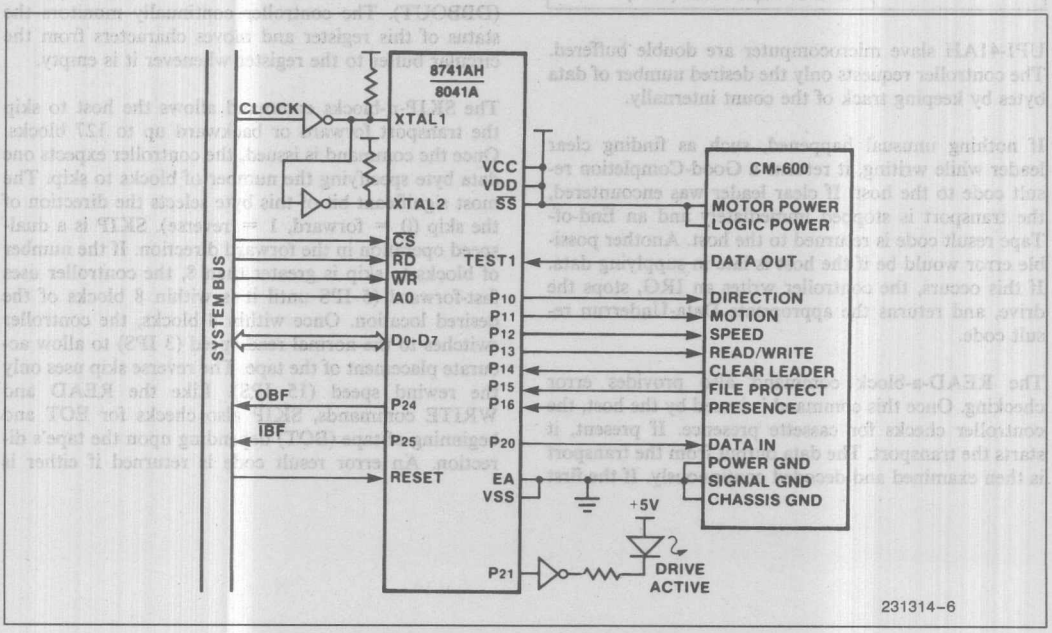


Figure 6. Controller/Transport System Schematic

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A0	Register
O	O	I	O	DBBOUT
O	O	I	I	STATUS
O	I	O	O	DBBIN (DATA)
O	I	O	I	DBBIN (COMMAND)
I	X	X	X	NONE

Figure 7. 8741AH/8041A Interface Register Decoding

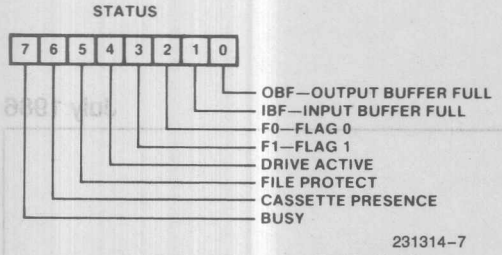


Figure 8. Status Register Bit Definition

The Status register contains flags which give the host the status of various operations within the controller. Its format is given in Figure 8. The Input Buffer Full ($\overline{IBF}$) and Output Buffer Full (OBF) flags show the Status of the DBBIN and DBBOUT registers respectively. $\overline{IBF}$ indicates when the DBBIN register contains data written by the host. The host may write to DBBIN only when $\overline{IBF}$ is 0. Likewise, the host may read DBBOUT only when OBF is set to a 1. These bits are handled automatically by the UPI-41AH internal hardware. FLAG 0 (F_0) and FLAG 1 (F_1) are general purpose flags used internally by the controller which have no meaning externally.

The remaining four bits are user-definable. For this application they are DRIVE ACTIVE, FILE PROTECT, CASSETTE PRESENCE, and BUSY flags. The FILE PROTECT and CASSETTE PRESENCE flags reflect the state of the corresponding I/O lines from the transport. DRIVE ACTIVE is set whenever the transport motor is on and the controller is performing an operation. The BUSY flag indicates whether the contents of the DBBOUT register is data or a result code. The BUSY flag is set whenever a command is issued by the host and accepted by the controller. As long as BUSY is set, any character found in DBBOUT is a result code. Thus whenever the host finds OBF set, it should test the BUSY flag to determine whether the character is data or a result code.

Notice the OBF and $\overline{IBF}$ are available as interrupt outputs to the host processor, Figure 6. These outputs are self-clearing, that is, OBF is set automatically upon the controller loading DBBOUT and cleared automatically by the host reading DBBOUT. Likewise $\overline{IBF}$ is cleared to a 0 by the host writing into DBBIN: set to a 1 when the controller reads DBBIN into the accumulator.

The flow charts of Figure 9 show the flow of sample host software assuming a polling software interface between the host and the controller. The WRITE command requires two additional count bytes which form the 16-bit byte count. These extra bytes are "handshaked" into the controller using the IBF flag in the STATUS register. Once these bytes are written, the host writes data in response to IBF being cleared. This continues until the host finds OBF set indicating that the operation is complete and reads the result code from DBBOUT. No testing of BUSY is needed since only the result code appears in the DBBOUT register.

The READ command does require that BUSY be tested. Once the READ command is written into the controller, the host must test BUSY whenever OBF is set to determine whether the contents of DBBOUT is data from the tape or the result code.

THE CONTROLLER SOFTWARE

The UPI-41AH software to control the cassette can be divided up into various commands such as WRITE, READ and ABORT. In a previous version of this application note (May 1980), software was described that implemented these commands. This code however did not adequately compensate for speed variations of the motor during record and playback nor for data distortion caused by the magnetic media. Since then, a new code has been written to include these effects. This revised software is now available through the INTEL User's Library, INSITE. For more information on this software or INSITE, contact your local INTEL Sales Office.

July 1986

UPI-452 Accelerates iAPX 286 Bus Performance

CHRISTOPHER SCOTT
TECHNICAL MARKETING ENGINEER
INTEL CORPORATION

Order Number: 292018-001

INTRODUCTION

The UPI-452 targets the leading problem in peripheral to host interfacing, the interface of a slow peripheral with a fast Host or "bus utilization". The solution is data buffering to reduce the delay and overhead of transferring data between the Host microprocessor and I/O subsystem. The Intel CMOS UPI-452 solves this problem by combining a sophisticated programmable FIFO buffer and a slave interface with an MSC-51 based microcontroller.

The UPI-452 is Intel's newest Universal Peripheral Interface family member. The UPI-452 FIFO buffer enables Host—peripheral communications to be through streams or bursts of data rather than by individual bytes. In addition the FIFO provides a means of embedding commands within a stream or block of data. This enables the system designer to manage data and commands to further off-load the Host.

The UPI-452 interfaces to the iAPX 286 microprocessor as a standard Intel slave peripheral device. READ, WRITE, CS and address lines from the Host are used to access all of the Host addressable UPI-452 Special Function Registers (SFR).

The UPI-452 combines an MSC-51 microcontroller, with 256 bytes of on-chip RAM and 8K bytes of EPROM/ROM, twice that of the 80C51, a two channel DMA controller and a sophisticated 128 byte, two channel, bidirectional FIFO in a single device. The UPI-452 retains all of the 80C51 architecture, and is fully compatible with the MSC-51 instruction set.

This application note is a description of an iAPX 286 to UPI-452 slave interface. Included is a discussion of the respective timings and design considerations. This application note is meant as a supplement to the UPI-452 Advance Data Sheet. The user should consult the data sheet for additional details on the various UPI-452 functions and features.

UPI-452 iAPX 286 SYSTEM CONFIGURATION

The interface described in this application note is shown in Figure 1, iAPX 286 UPI-452 System Block Diagram. The iAPX 286 system is configured in a local bus architecture design. DMA between the Host and the UPI-452 is supported by the 82258 Advanced DMA Controller. The Host microprocessor accesses all UPI-452 externally addressable registers through address decoding (see Table 3, UPI-452 External Address Decoding). The timings and interface descriptions below are given in equation form with examples of specific calculations. The goal of this application note is a set of interface analysis equations. These equations are the tools a system designer can use to fully utilize the features of the UPI-452 to achieve maximum system performance.

HOST-UPI-452 FIFO SLAVE INTERFACE

The UPI-452 FIFO acts as a buffer between the external Host 80286 and the internal CPU. The FIFO allows the Host - peripheral interface to achieve maximum decoupling of the interface. Each of the two FIFO channels is fully user programmable. The FIFO buffer ensures that the respective CPU, Host or internal CPU, receives data in the same order as transmitted. Three slave bus interface handshake methods are supported by the UPI-452; DMA, Interrupt and Polled.

The interface between the Host 80286 and the UPI-452 is accomplished with a minimum of signals. The 8 bit data bus plus READ, WRITE, CS, and A0-2 provide access to all of the externally addressable UPI-452 registers including the two FIFO channels. Interrupt and DMA handshaking pins are tied directly to the interrupt controller and DMA controller respectively.

DMA transfers between the Host and UPI-452 are controlled by the Host processors DMA controller. In the example shown in Figure 1, the Host DMA controller is the 82258 Advanced DMA Controller. An internal DMA transfer to or from the FIFO, as well as between other internal elements, is controlled by the UPI-452 internal DMA processor. The internal DMA processor can also transfer data between Input and Output FIFO channels directly. The description that follows details the UPI-452 interface from both the Host processor's and the UPI-452's internal CPU perspective.

One of the unique features of the UPI-452 FIFO is its ability to distinguish between commands and data embedded in the same data block. Both interrupts and status flags are provided to support this operation in either direction of data transfer. These flags and interrupts are triggered by the FIFO logic independent of, and transparent to either the Host or internal CPUs. Commands embedded in the data block, or stream, are called Data Stream Commands.

Programmable FIFO channel Thresholds are another unique feature of the UPI-452. The Thresholds provide for interrupting the Host only when the Threshold number of bytes can be read or written to the FIFO buffer. This further decouples the Host UPI-452 interface by relieving the Host of polling the buffer to determine the number of bytes that can be read or written. It also reduces the chances of overrun and underrun errors which must be processed.

The UPI-452 also provides a means of bypassing the FIFO, in both directions, for an immediate interrupt of either the Host or internal CPU. These commands are called Immediate Commands. A complete description of the internal FIFO logic operation is given in the FIFO Data Structure section.

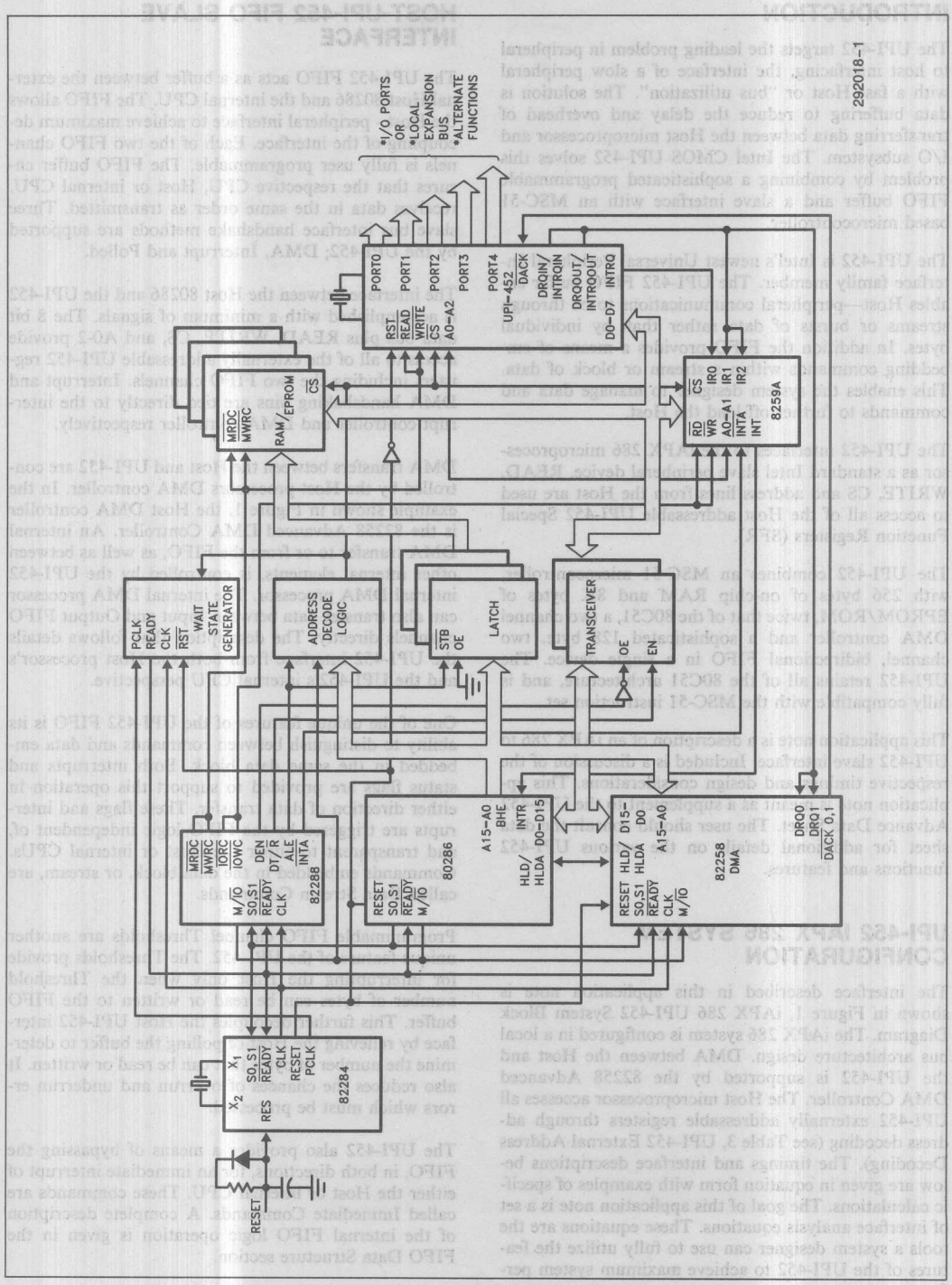


Figure 1. iAPX 286 UPI-452 System Block Diagram

UPI-452 INITIALIZATION

The UPI-452 at power-on reset automatically performs a minimum initialization of itself. The UPI-452 notifies the Host that it is in the process of initialization by setting a Host Status SFR bit. The user UPI-452 program must release the UPI-452 from initialization for the FIFO to be accessible by the Host. This is the minimum Host to UPI-452 initialization sequence. All further initialization and configuration of the UPI-452, including the FIFO, is done by the internal CPU under user program control. No interaction or programming is required by the Host 80286 for UPI-452 initialization.

At power-on reset the UPI-452 automatically enters FIFO DMA Freeze Mode by resetting the Slave Control (SLCON) SFR FIFO DMA Freeze/Normal Mode bit to FIFO DMA Freeze Mode (FRZ = "0"). This forces the Slave Status (SSTAT) and Host Status (HSTAT) SFR FIFO DMA Freeze/Normal Mode bits to FIFO DMA Freeze Mode In Progress. FIFO DMA Freeze Mode allows the FIFO interface to be configured, by the internal CPU, while inhibiting Host access to the FIFO.

The MODE SFR is forced to zero at reset. This disables, (tri-states) the DRQIN/INTRQIN, DRQOUT/INTRQOUT and INTRQ output pins. INTRQ is inhibited from going active to reflect the fact that a Host Status SFR bit, FIFO DMA Freeze Mode, is active. If the MODE SFR INTRQ configure bit is enabled (='1'), before the Slave Control and Host Status SFR FIFO DMA Freeze/Normal Mode bit is set to Normal Mode, INTRQ will go active immediately.

The first action by the Host following reset is to read the UPI-452 Host Status SFR Freeze/Normal Mode bit to determine the status of the interface. This may be done in response to a UPI-452 INTRQ interrupt, or by polling the Host Status SFR. Reading the Host Status SFR resets the INTRQ line low.

Any of the five FIFO interface SFRs, as well as a variety of additional features, may be programmed by the internal CPU following reset. At power-on reset, the five FIFO Special Function Registers are set to their default values as listed in Table 1. All reserved location bits are set to one, all other bits are set to zero in these three SFRs. The FIFO SFRs listed in Table 1 can be programmed only while the UPI-452 is in FIFO DMA Freeze Mode. The balance of the UPI-452 SFRs default values and descriptions are listed in the UPI-452 Advance Data Sheet in the Intel Microsystems Component Handbook Volume II and Microcontroller Handbook.

The above sequence is the minimum UPI-452 internal initialization required. The last initialization instruction must always set the UPI-452 to Normal Mode. This causes the UPI-452 to exit Freeze Mode and enables Host read/write access of the FIFO. The internal CPU sets the Slave Control (SLCON) SFR FIFO DMA Freeze/Normal Mode (FRZ) bit high (= 1) to activate Normal Mode. Ths causes the Slave Status (SSTAT) and Host Status (HSTAT) SFR FIFO DMA Freeze Mode bits to be set to Normal Mode. Table 2, UPI-452 Initialization Event Sequence Example, shows a summary of the initialization events described above.

Table 1. FIFO Special Function Register Default Values

SFR Name	Label	Reset Value
Channel Boundary Pointer	CBP	40H/64D
Output Channel Read Pointer	ORPR	40H/64D
Output Channel Write Pointer	OWPR	40H/64D
Input Channel Read Pointer	IRPR	00H/0D
Input Channel Write Pointer	IWPR	00H/0D
Input Threshold	ITH	00H/0D
Output Threshold	OTH	01H/1D

Table 2. UPI-452 Initialization Event Sequence Example

Event Description	SFR/bit
Power-on Reset	
UPI-452 forces FIFO DMA Freeze Mode (Host access to FIFO inhibited)	SLCON FRZ = 0
UPI-452 forces Slave Status and Host Status SFR to FIFO DMA Freeze Mode In Progress	SSTAT SST5 = 0 HSTAT HST1 = 1
UPI-452 forces all SFRs, including FIFO SFRs, to default values.	
* UPI-452 user program enables INTRQ, INTRQ goes active, high	MODE MD4 = 1
* Host $\overline{\text{READ}}$'s UPI-452 Host Status (HSTAT) SFR to determine interrupt source, INTRQ goes low	
* UPI-452 user program initializes any other SFRs; FIFO, Interrupts, Timers/Counters, etc.	
User program sets Slave Control SFR to Normal Mode (Host access to FIFO enabled)	SLCON FRZ = 1
UPI-452 forces Slave and Host Status SFRs bits to Normal Operation	SSTAT SST5 = 1 HSTAT HST1 = 0
* Host polls Host Status SFR to determine when it can access the FIFO - or - * Host waits for UPI-452 Request for Service interrupt to access FIFO	

* user option

FIFO DATA STRUCTURES

Overview

The UPI-452 provides three means of communication between the Host microprocessor and the UPI-452 in either direction;

Data
Data Stream Commands
Immediate Commands

Data and Data Stream Commands (DSC) are transferred between the Host and UPI-452 through the UPI-452 FIFO buffer. The third, Immediate Commands, provides a means of bypassing the FIFO entirely. These three data types are in addition to direct access by either Host or Internal CPU of dedicated Status and Control Special Function Registers (SFR).

The FIFO appears to both the Host 80286 and the internal CPU as 8 bits wide. Internally the FIFO is logically nine bits wide. The ninth bit indicates whether the byte is a data or a Data Stream Command (DSC) byte; 0 = data, 1 = DSC. The ninth bit is set by the FIFO logic in response to the address specified when writing to the FIFO by either Host or internal CPU. The FIFO uses the ninth bit to condition the UPI-452 interrupts and status flags as a byte is made available for a Host or internal CPU read from the FIFO. Figures 2 and 3 show the structure of each FIFO channel and the logical ninth bit.

It is important to note that both data and DSCs are actually entered into the FIFO buffer (see Figures 2 and 3). External addressing of the FIFO determines the state of the internal FIFO logic ninth bit. Table 3 shows the UPI-452 External Address Decoding used by the Host and the corresponding action. The internal CPU interface to the FIFO is essentially identical to the external Host interface. Dedicated internal Special Function Registers provide the interface between the FIFO, internal CPU and the internal two channel DMA processor. FIFO read and write operations by the Host and internal CPU are interleaved by the UPI-452 so they appear to be occurring simultaneously.

The ninth bit provides a means of supporting two data types within the FIFO buffer. This feature enables the Host and UPI-452 to transfer both commands and data while maintaining the decoupled interface a FIFO buffer provides. The logical ninth bit provides both a means of embedding commands within a block of data and a means for the internal CPU, or external Host, to discriminate between data and commands. Data or DSCs may be written in any order desired. Data Stream Commands can be used to structure or dispatch the data by defining the start and end of data blocks or packets, or how the data following a DSC is to be processed.

A Data Stream Command (DSC) acts as an internal service routine vector. The DSC generates an interrupt to a service routine which reads the DSC. The DSC byte acts as an address vector to a user defined service routine. The address can be any program or data memory location with no restriction on the number of DSCs or address boundaries.

A Data Stream Command (DSC) can also be used to clear data from the FIFO or "FLUSH" the FIFO. This is done by appending a DSC to the end of a block of data entered in the FIFO which is less than the programmed threshold number of bytes. The DSC will cause an interrupt, if enabled, to the respective receiving CPU. This ensures that a less than Threshold number of bytes in the FIFO will be read. Two conditions force a Request for Service interrupt, if enabled, to the Host. The first is due to a Threshold number of bytes having been written to the FIFO Output channel; the second is if a DSC is written to the Output FIFO channel. If less than the Threshold number of bytes are written to the Output FIFO channel, the Host Status SFR flag will not be set, and a Request for Service interrupt will not be generated, if enabled. By appending a DSC to end of the data block, the FIFO Request for Service flag and/or interrupt will be generated.

An example of a FIFO Flush application is a mass storage subsystem. The UPI-452 provides the system interface to a subsystem which supports tape and disk storage. The FIFO size is dynamically changed to provide the maximum buffer size for the direction of transfer. Large data blocks are the norm in this application. The FIFO Flush provides a means of purging the FIFO of the last bytes of a transfer. This guarantees that the block, no matter what its size, will be transmitted out of the FIFO.

Immediate Commands allow more direct communication between the Host processor and the UPI-452 by bypassing the FIFO in either direction. The Immediate Command IN and OUT SFRs are two more unique address locations externally and internally addressable. Both DSCs and Immediate Commands have internal interrupts and interrupt priorities associated with their operation. The interrupts are enabled or disabled by setting corresponding bits in the Slave Control (SLCON), Interrupt Enable (IEC), Interrupt Priority (IPC) and Interrupt Enable and Priority (IEP) SFRs. A detailed description of each of these may be found in the UPI-452 Advance Information Data Sheet.

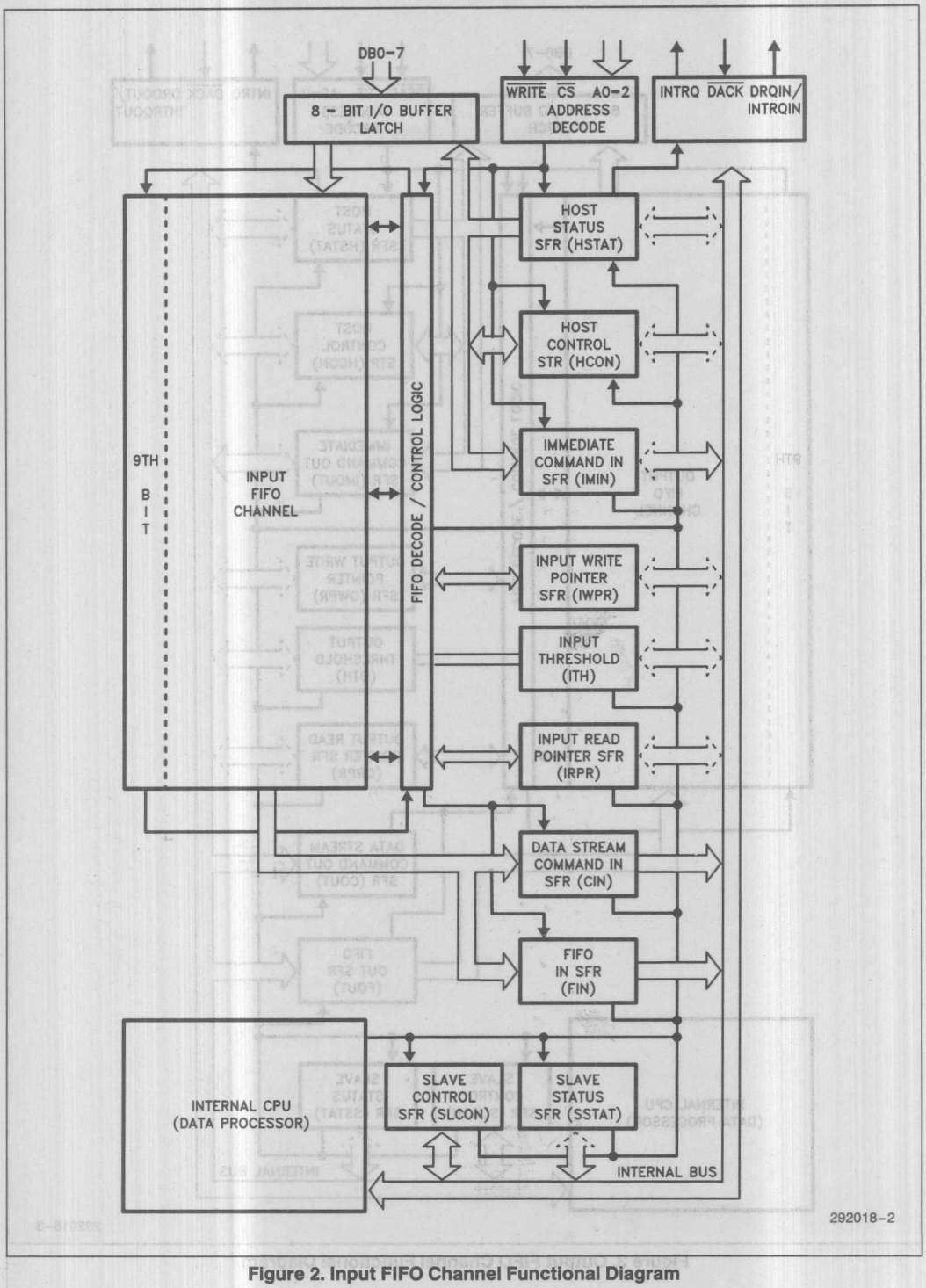


Figure 2. Input FIFO Channel Functional Diagram

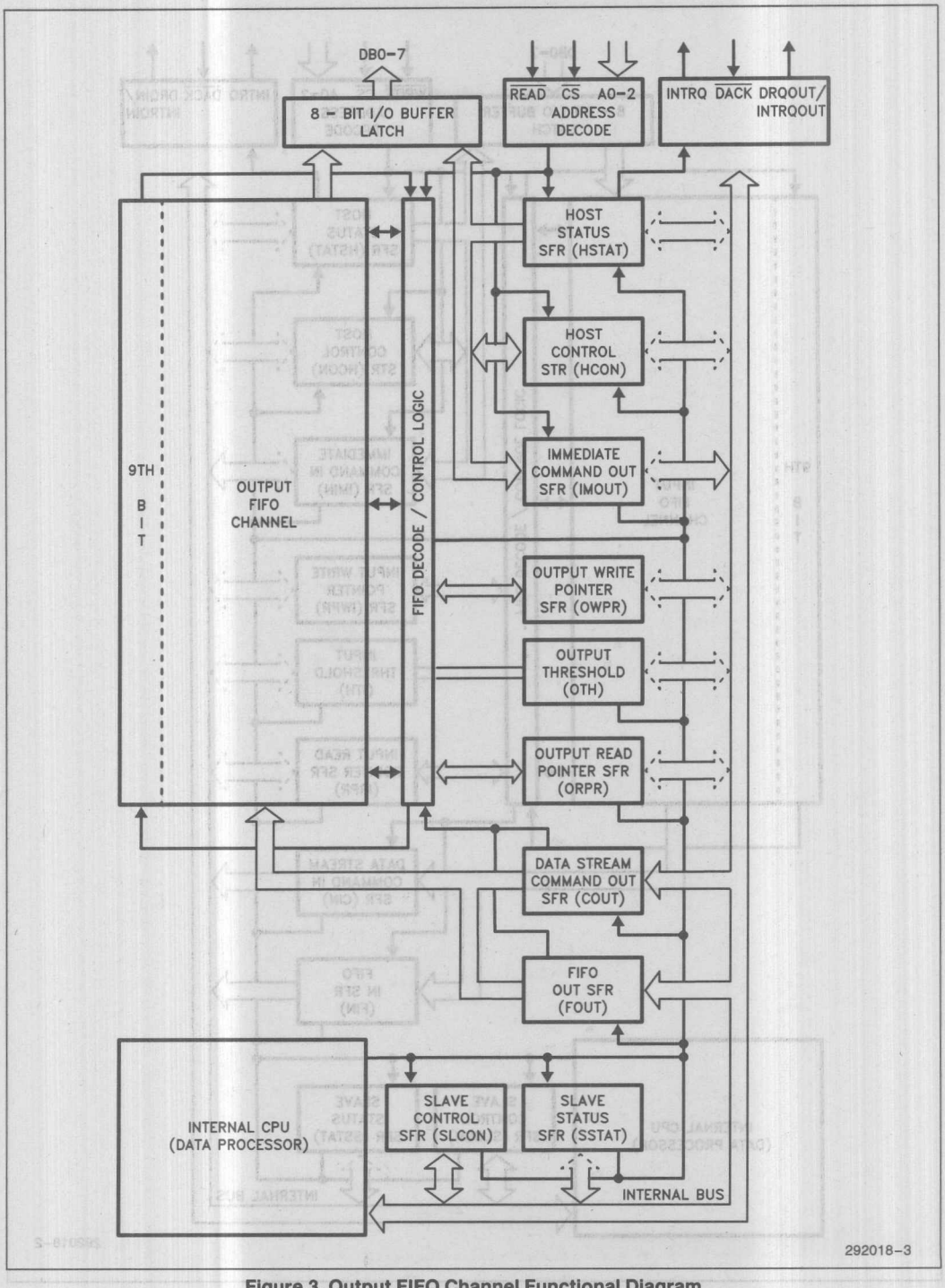


Figure 3. Output FIFO Channel Functional Diagram

Table 3. UPI-452 External Address Decoding

DACK	CS	A2	A1	A0	READ	WRITE
1	1	X	X	X	No Operation	No Operation
1	0	0	0	0	Data or DMA from Output FIFO Channel	Data or DMA to Input FIFO Channel
1	0	0	0	1	Data Stream Command from Output FIFO Channel	Data Stream Command to Input FIFO Channel
1	0	0	1	0	Host Status SFR Read	Reserved
1	0	0	1	1	Host Control SFR Read	Host Control SFR Write
1	0	1	0	0	Immediate Command SFR Read	Immediate Command SFR Write
1	0	1	1	X	Reserved	Reserved
0	X	X	X	X	DMA Data from Output FIFO Channel	DMA Data to Input FIFO Channel

Below is a detailed description of each FIFO channel's operation, including the FIFO logic response to the ninth bit, as a byte moves through the channel. The description covers each of the three data types for each channel. The details below provide a picture of the various FIFO features and operation. By understanding the FIFO structure and operation the user can optimize the interface to meet the requirements of an individual design.

OUTPUT CHANNEL

This section covers the data path from the internal CPU to the HOST. Data Stream Command or Immediate Command processing during Host DMA Operations is covered in the DMA section.

UPI-452 Internal Write to the FIFO

The internal CPU writes data and Data Stream Commands into the FIFO through the FIFO OUT (FOUT) and Command OUT (COUT) SFRs. When a Threshold number of bytes has been written, the Host Status SFR Output FIFO Request for Service bit is set and an interrupt, if enabled, is generated to the Host. Either the INTRQ or DRQOUT/INTRQOUT output pins can be used for this interrupt as determined by the MODE and Host Control (HCON) SFR setting. The Host responds to the Request for Service interrupt by reading the Host Status (HSTAT) SFR to determine the source of the interrupt. The Host then reads the Threshold number of bytes from the FIFO. The internal CPU may continue to write to the FIFO during the Host read of the FIFO Output channel.

Data Stream Commands may be written to the Output FIFO channel at any time during a write of data bytes. The write instruction need only specify the Command Out (COUT) SFR in the direct register instruction used. Immediate Commands may also be written at any time to the Immediate Command OUT (IMOUT) SFR. The Host reads Immediate Commands from the Immediate Command OUT (IMOUT).

The internal CPU can determine the number of bytes to write to the FIFO Output channel in one of three ways. The first, and most efficient, is by utilizing the internal DMA processor which will automatically manage the writing of data to avoid Underrun or Overrun Errors. The second is for the internal CPU to read the Output FIFO channels Read and Write Pointers and compare their values to determine the available space. The third method for determining the available FIFO space is to always write the programmed channel size number of bytes to the Output FIFO. This method would use the Overrun Error flag and interrupt to halt FIFO writing whenever the available space was less than the channel size. The interrupt service routine could read the channel pointers to determine or monitor the available channel space. The time required for the internal CPU to write data to the Output FIFO channel is a function of the individual instruction cycle time and the number of bytes to be written.

Host Read from the FIFO

The Host reads data or Data Stream Commands (DSC) from the FIFO in response to the Host Status (HSTAT) SFR flags and interrupts, if enabled. All Host read operations access the same UPI-452 internal I/O Buffer Latch. At the end of the previous Host FIFO read cycle a byte is loaded from the FIFO into the I/O Buffer Latch and Host Status (HSTAT) SFR bit 5 is set or cleared (1 = DSC, 0 = data) to reflect the state of the byte's FIFO ninth bit. If the FIFO ninth bit is set (= 1) indicating a DSC, an interrupt is generated to the external Host via INTRQ pin or INTRQIN/INTRQOUT pins as determined by Host Control (HCON) SFR bit 1. The Host then reads the Host Status (HSTAT) SFR to determine the source of the interrupt.

The most efficient Host read operation of the FIFO Output channel is through the use of Host DMA. The UPI-452 fully supports external DMA handshaking. The MODE and Host Control SFRs control the configuration of UPI-452 Host DMA handshake outputs. If Host DMA is used the Threshold Request for Service interrupt asserts the UPI-452 DMA Request (DRQOUT) output. The Host DMA processor acknowledges with $\overline{\text{DACK}}$ which acts as a chip select of the FIFO channels. The DMA transfer would stop when either the threshold byte count had been read, as programmed in the Host DMA processor, or when the DRQOUT output is brought inactive by the UPI-452.

INPUT CHANNEL

This section covers the data path from the HOST to the internal CPU or internal DMA processor. The details of Data Stream Command or Immediate Command processing during internal DMA operations are covered in the DMA section below.

Host Write to the FIFO

The Host writes data and Data Stream Commands into the FIFO through the FIFO IN (FIN) and Command IN (CIN) SFRs. When a Threshold number of bytes has been read out of the Input FIFO channel by the internal CPU, the Host Status SFR Input FIFO Request for Service bit is set and an interrupt, if enabled, is generated to the Host. The Input FIFO Threshold interrupt tells the Host that it may write the next block of data into the FIFO. Either the INTRQ or DRQIN/INTRQIN output pins can be used for this interrupt as determined by the MODE and Host Control (HCON) SFR settings. The Host may continue to write to the FIFO Input channel during the internal CPU read of the FIFO. Data Stream Commands may be written to the FIFO Input channel at any time during a write of data bytes. Immediate Commands may also be written at any time to the Immediate Command IN (IMIN) SFR.

The Host also has three methods for determining the available FIFO space. Two are essentially identical to that of the internal CPU. They involve reading the FIFO Input channel pointers and using the Host Status SFR Underrun and Overrun Error flags and Request for Service interrupts these would generate, if enabled in combination. The third involves using the UPI-452 Host DMA controller handshake signals and the programmed Input FIFO Threshold. The Host would receive a Request for Service interrupt when an Input FIFO channel has a Theshold number of bytes able to be written by the Host. The Host service routine would then write the Threshold number of bytes to the FIFO.

If a Host DMA is used to write to the FIFO Input channel, the Threshold Request for Service interrupt could assert the UPI-452 DRQIN output. The Host DMA processor would assert $\overline{\text{DACK}}$ and the FIFO write would be completed by Host the DMA processor. The DMA transfer would stop when either the Threshold byte count had been written or the DRQIN output was removed by the UPI-452. Additional details on Host and internal DMA operation is given below.

Internal Read of the FIFO

At the end of an internal CPU read cycle a byte is loaded from the FIFO buffer into the FIFO IN/Command IN SFR and Slave Status (SSTAT) SFR bit 1 is set or cleared (1 = data, 0 = DSC) to reflect the state of the FIFO ninth bit. If the byte is a DSC, the FIFO ninth bit is set (= 1) and an interrupt is generated, if enabled, to the Internal CPU. The internal CPU then reads the Slave Status (SSTAT) SFR to determine the source of the interrupt.

Immediate Commands are written by the Host and read by the internal CPU through the Immediate Command IN (IMIN) SFR. Once written, an Immediate Command sets the Slave Status (SSTAT) SFR flag bit and generates an interrupt, if enabled, to the internal CPU. In response to the interrupt the internal CPU reads the Slave Status (SSTAT) SFR to determine the source of the interrupt and service the Immediate Command.

FIFO INPUT/OUTPUT CHANNEL SIZE

Host

The Host does not have direct control of the FIFO Input or Output channel sizes or configuration. The Host can, however, issue Data Stream Commands or Immediate Commands to the UPI-452 instructing the UPI-452 to reconfigure the FIFO interface by invoking FIFO DMA Freeze Mode. The Data Stream Command or Immediate Command would be a vector to a service routine which performs the specific reconfiguration.

UPI-452 Internal

The default power-on reset FIFO channel sizes are listed in the "Initialization" section and can be set only by the internal CPU during FIFO DMA Freeze Mode. The FIFO channel size is selected to achieve the optimum application performance. The entire 128 byte FIFO can be allocated to either the Input or Output channel. In this case the other channel consists of a single SFR; FIFO IN/Command IN or FIFO OUT/Command OUT SFR. Figure 4 shows a FIFO division with a portion devoted to each channel. Figure 5 shows a FIFO configuration with all 128 bytes assigned to the Output channel.

The FIFO channel Threshold feature allows the user to match the FIFO channel size and the performance of the internal and Host data transfer rates. The programmed Threshold provides an elasticity to the data transfer operation. An example is if the Host FIFO

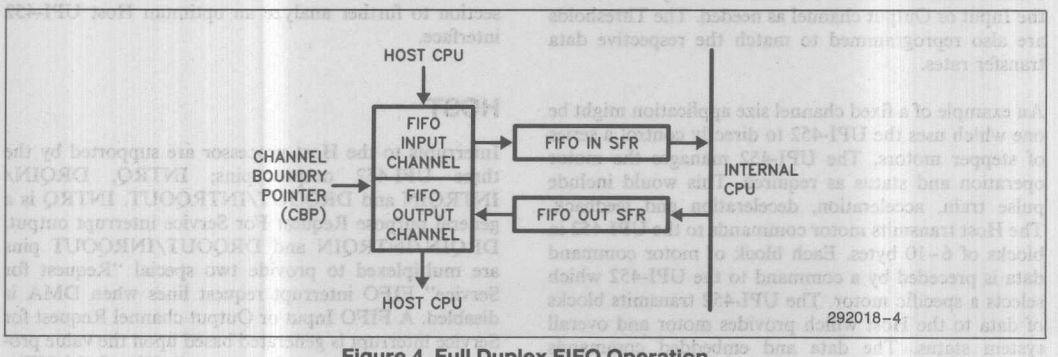


Figure 4. Full Duplex FIFO Operation

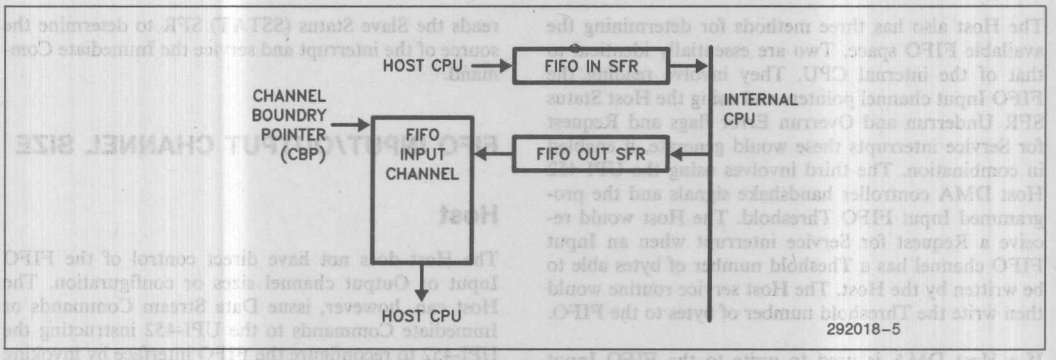


Figure 5. Entire FIFO Assigned to Output Channel

data transfer rate is twice as fast as the internal FIFO DMA data transfer rate. In this example the FIFO Input channel size is programmed to be 64 bytes and the Input channel Threshold is programmed to be 20 bytes. The Host writes the first 64 bytes to the Input FIFO. When the internal DMA processor has read 20 bytes the Threshold interrupt, or DMA request (DRQIN), is generated to signal the Host to begin writing more data to the Input FIFO channel. The internal DMA processor continues to read data from the Input channel as the Host, or Host DMA processor, writes to the FIFO. The Host can write 40 bytes to the FIFO Input channels in the time it takes for the internal DMA processor to read 20 more bytes from it. This will keep both the Host and internal DMA operating at their maximum rates without forcing one to wait for the other.

Two methods of managing the FIFO size are possible; fixed and variable channel size. A fixed channel size is one where the channel is configured at initialization and remains unchanged throughout program execution. In a variable FIFO channel size, the configuration is changed dynamically to meet the data transmission requirements as needed. An example of a variable channel size application is the mass storage subsystem described earlier. To meet the demands of a large data block transfer the FIFO size could be fully allocated to the Input or Output channel as needed. The Thresholds are also reprogrammed to match the respective data transfer rates.

An example of a fixed channel size application might be one which uses the UPI-452 to directly control a series of stepper motors. The UPI-452 manages the motor operation and status as required. This would include pulse train, acceleration, deceleration and feedback. The Host transmits motor commands to the UPI-452 in blocks of 6–10 bytes. Each block of motor command data is preceded by a command to the UPI-452 which selects a specific motor. The UPI-452 transmits blocks of data to the Host which provides motor and overall system status. The data and embedded commands structure, indicating the specific motor, is the same. In this example the default 64 bytes per channel might be adequate for both channels.

INTERRUPT RESPONSE TIMING

Interrupts enable the Host UPI-452 FIFO buffer interface and the internal CPU FIFO buffer interface to operate with a minimum of overhead on the respective CPU. Each CPU is "interrupted" to service the FIFO on an as needed basis only. In configuring the FIFO buffer Thresholds and choosing the appropriate internal DMA Mode the user must take into account the interrupt response time for both CPUs. These response times will affect the DMA transfer rates for each channel. By choosing FIFO channel Thresholds which reflect both the respective DMA transfer rate and the interrupt response time the user will achieve the maximum data throughput and system bus decoupling. This in turn will mean the overall available system bus bandwidth will increase.

The following section describes the FIFO interrupt interface to the Host and internal CPU. It also describes an analysis of sample interrupt response times for the Host and UPI-452 internal CPU. These equations and the example times shown are then used in the DMA section to further analyze an optimum Host UPI-452 interface.

HOST

Interrupts to the Host processor are supported by the three UPI-452 output pins; INTRQ, DRQIN/INTRQIN and DRQOUT/INTRQOUT. INTRQ is a general purpose Request For Service interrupt output. DRQIN/INTRQIN and DRQOUT/INRQOUT pins are multiplexed to provide two special "Request for Service" FIFO interrupt request lines when DMA is disabled. A FIFO Input or Output channel Request for Service interrupt is generated based upon the value programmed in the respective channel's Threshold SFRs; Input Threshold (ITHR), and Output Threshold

(OTHR) SFRs. Additional interrupts are provided for FIFO Underrun and Overrun Errors, Data Stream Commands, and Immediate Commands. Table 4 lists the eight UPI-452 interrupt sources as they appear in the HSTAT SFR to the Host processor.

Table 4. UPI-452 to Host Interrupt Sources

HSTAT SFR Bit	Interrupt Source
HST7	Output FIFO Underrun Error
HST6	Immediate Command Out SFR Status
HST5	Data Stream Command/Data at Output FIFO Status
HST4	Output FIFO Request for Service Status
HST3	Input FIFO Overrun Error Condition
HST2	Immediate Comamnd In SFR Status
HST1	FIFO DMA Freeze/Normal Mode Status
HST0	Input FIFO Request for Service

The interrupt response time required by the Host processor is application and system specific. In general, a typical sequence of Host interrupt response events and the approximate times associated with each are listed in Equation 1.

The example assumes the hardware configuration shown in Figure 1, iAPX 286/UPI-452 Block Diagram, with an 8259A Programmable Interrupt Controller. The timing analysis in Equation 1 also assumes the following; no other interrupt is either in process or pending, nor is the 286 in a LOCK condition. The current instruction completion time is 8 clock cycles (800 ns @ 10 MHz), or 4 bus cycles. The interrupt service routine first executes a PUSHA instruction, PUSH All General Registers, to save all iAPX 286 internal registers. This requires 19 clocks (or 2.0 μs @ 10 MHz), or 10 bus cycles (rounded to complete bus cycle). The next service routine instruction reads the UPI-452 Host Status SFR to determine the interrupt source.

It is important to note that any UPI-452 INTRQ interrupt service routine should ALWAYS mask for the Freeze Mode bit first. This will insure that Freeze Mode always has the highest priority. This will also save the time required to mask for bits which are forced inactive during Freeze Mode, before checking the Freeze Mode bit. Access to the FIFO channels by the Host is inhibited during Freeze Mode. Freeze Mode is covered in more detail below.

To initiate the interrupt the UPI-452 activates the INTRQ output. The interrupt acknowledge sequence requires two bus cycles, 400 ns (10 MHz iAPX 286), for the two INTA pulse sequence.

Equation 1. Host Interrupt Response Time

Action	Time	Bus Cycles*
Current instruction execution completion	800 ns	4
INTA sequence	400 ns	2
Interrupt service routine (time to host first $\overline{\text{READ}}$ of UPI-452)	2000 ns	10
Total Interrupt Response Time	2.3 μs	16

NOTE:
10 MHz iAPX 286 bus cycle, 200 ns each

UPI-452 Internal

The internal CPU FIFO interrupt interface is essentially identical to that of the Host to the FIFO. Three internal interrupt sources support the FIFO operation; FIFO-Slave bus Interface Buffer, DMA Channel 0 and DMA Channel 1 Requests. These interrupts provide a maximum decoupling of the FIFO buffer and the internal CPU. The four different internal DMA Modes available add flexibility to the interface.

The FIFO-Slave Bus Interface interrupt response is also similar to the Host response to an INTRQ Request for Service interrupt. The internal CPU responds to the interrupt by reading the Slave Status (SSTAT) SFR to determine the source of the interrupt. This allows the user to prioritize the Slave Status flag response to meet the users application needs.

The internal interrupt response time is dependent on the current instruction execution, whether the interrupt is enabled, and the interrupt priority. In general, to finish execution of the current instruction, respond to the interrupt request, push the Program Counter (PC) and vector to the first instruction of the interrupt service routine requires from 38 to 86 oscillator periods (2.38 to 5.38 μs @ 16 MHz). If the interrupt is due to an Immediate Command or DSC, additional time is required to read the Immediate Command or DSC SFR and vector to the appropriate service routine. This means two service routines back to back. One service routine to read the Slave Status (SSTAT) SFR to determine the source of the Request for Service interrupt, and second the service routine pointed to by the Immediate Command or DSC byte read from the respective SFR.

DMA

DMA is the fastest and most efficient way for the Host or internal CPU to communicate with the FIFO buffer. The UPI-452 provides support for both of these DMA paths. The two DMA paths and operations are fully independent of each other and can function simultaneously. While the Host DMA processor is performing a DMA transfer to or from the FIFO, the UPI-452 internal DMA processor can be doing the same.

Below are descriptions of both the Host and internal DMA operations. Both DMA paths can operate asynchronously and at different transfer rates. In order to make the most of this simultaneous asynchronous operation it is necessary to calculate the two transfer rates and accurately match their operations. Matching the different transfer rates is done by a combination of accurately programmed FIFO channel size and channel Thresholds. This provides the maximum Host and internal CPU to FIFO buffer interface decoupling. Below is a description of each of the two DMA operations and sample calculations for determining transfer rates. The next section of this application note, "Interface Latency", details the considerations involved in analyzing effective transfer rates when the overhead associated with each transfer is considered.

HOST FIFO DMA

DMA transfers between the Host and UPI-452 FIFO buffer are controlled by the Host CPU's DMA controller, and is independent of the UPI-452's internal two channel DMA processor. The UPI-452's internal DMA processor supports data transfers between the UPI-452 internal RAM, external RAM (via the Local Expansion Bus) and the various Special Function Registers including the FIFO Input and Output channel SFRs.

The maximum DMA transfer rate is achieved by the minimum DMA transfer cycle time to accomplish a source to destination move. The minimum Host UPI-452 FIFO DMA cycle time possible is determined by the $\overline{\text{READ}}$ and $\overline{\text{WRITE}}$ pulse widths, UPI-452 command recovery times in relation to the DMA transfer timing and DMA controller transfer mode used. Table 5 shows the relationship between the iAPX-286, iAPX-186 and UPI-452 for various DMA as well as non-DMA byte by byte transfer modes versus processor frequencies.

Host processor speed vs wait states required with UPI-452 running at 16 MHz:

Table 5. Host UPI-452 Data Transfer Performance

Processor & Speed	Wait States: Back to Back $\overline{\text{READ}}$/$\overline{\text{WRITE}}$'s	DMA: Single Cycle	Two Cycle
iAPX-186* 8 MHz	0	N/A*	0
10 MHz	0	N/A*	0
12.5 MHz	1	N/A*	0
iAPX-286** 6 MHz	0	0	0
8 MHz	1	1	0
10 MHz	2	2	0

NOTES:
* iAPX 186 On-chip DMA processor is two cycle operation only.
** iAPX 286 assumes 82258 ADMA (or other DMA) running 286 bus cycles at 286 clock rate.

In this application note system example, shown in Figure 1, DMA operation is assumed to be two bus cycle I/O to memory or memory to I/O. Two cycle DMA consists of a fetch bus cycle from the source and a store bus cycle to the destination. The data is stored in the DMA controller's registers before being sent to the destination. Single cycle DMA transfers involve a simultaneous fetch from the source and store to the destination. As the most common method of I/O-memory DMA operation, two cycle DMA transfers are the focus of this application note analysis. Equation 2 illustrates a calculation of the overall transfer rate between the FIFO buffer and external Host for a maximum FIFO size transfer. The equation does not account for the latency of initiating the DMA transfer.

Equation 2. Host FIFO DMA Transfer Rate—Input or Output Channel

2 Cycle DMA Transfer-I/O (UPI-452) to System Memory
= FIFO channel size* (DMA $\overline{\text{READ}}$/$\overline{\text{WRITE}}$ FIFO time + DMA $\overline{\text{WRITE}}$/$\overline{\text{READ}}$ Memory Time)
= 128 bytes* (200 ns + 200 ns)
= 51.2 μs
= 256 bus cycles*

NOTES:
*10 MHz iAPX 286, 200 ns bus cycles.

The UPI-452 design is optimized for high speed DMA transfers between the Host and the FIFO buffer. The

UPI-452 internal FIFO buffer control logic provides the necessary synchronization of the external Host event and the internal CPU machine cycle during UPI-452 $\overline{RD}/\overline{WR}$ accesses. This internal synchronization is addressed by the TCC AC specification of the UPI-452 shown in Figure 6. TCC is the time from the leading or trailing edge of a UPI-452 $\overline{RD}/\overline{WR}$ to the same edge of the next UPI-452 $\overline{RD}/\overline{WR}$. The TCC time is effectively another way of measuring the system bus cycle time with reference to UPI-452 accesses.

In the iAPX-286 10 MHz system depicted in this application note the bus cycle time is 200 ns. Alternate cycle accesses of the UPI-452 during two cycle DMA operation yields a TCC time of 400 ns which is more than the TCC minimum time of 375 ns. Back to back Host UPI-452 $\overline{READ}/\overline{WRITE}$ accesses may require wait states as shown in Table 5. The difference between 10 MHz iAPX-186 and 10 MHz iAPX 286 required wait states is due to the number of clock cycles in the respective bus cycle timings. The four clocks in a 10 MHz iAPX 186 bus cycle means a minimum TCC time of 400 ns versus 200 ns for a 10 MHz iAPX 286 with two clock cycle zero wait state bus cycle.

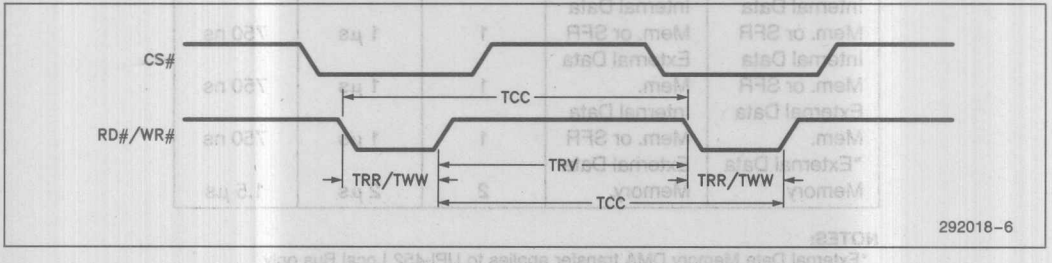
DMA handshaking between the Host DMA controller and the UPI-452 is supported by three pins on the UPI-452; DRQIN/INTRQIN, DRQOUT/INTRQOUT and $\overline{DACK}$. The DRQIN/INTRQIN and DRQOUT/INTRQOUT outputs are two multiplexed DMA or interrupt request pins. The function of these pins is controlled by MODE SFR bit 6 (MD6). DRQIN and DRQOUT provide a direct interface to the Host system DMA controller (see Figure 1). In response to a DRQIN or DRQOUT request, the Host DMA controller initiates control of the system bus using HLD/HLDA. The FIFO Input or Output channel transfer is accomplished with a minimum of Host overhead and system bus bandwidth.

The third handshake signal pin is $\overline{DACK}$ which is used as a chip select during DMA data transfers. The UPI-452 Host $\overline{READ}$ and $\overline{WRITE}$ input signals select the respective Input and Output FIFO channel during DMA transfers. The $\overline{CS}$ and address lines provide DMA acknowledge for processors with onboard DMA controllers which do not generate a $\overline{DACK}$ signal.

The iAPX 286 Block I/O Instructions provide an alternative to two cycle DMA data transfers with approximately the same data rate. The String Input and Output instructions (INS & OUTS) when combined with the Repeat (REP) prefix, modifies INS and OUTS to provide a means of transferring blocks of data between I/O and Memory. The data transfer rate using REP INS/OUTS instructions is calculated in the same way as two cycle DMA transfer times. Each $\overline{READ}$ or $\overline{WRITE}$ would be 200 ns in a 10 MHz iAPX 286 system. The maximum transfer rate possible is 2.5 MBytes/second. The Block I/O FIFO data transfer calculation is the same as that shown in Equation 2 for two cycle DMA data transfers including TCC timing effects.

FIFO Data Structure and Host DMA

During a Host DMA write to the FIFO, if a DSC is to be written, the DMA transfer is stopped, the DSC is written and the DMA restarted. During a Host DMA read from the FIFO, if a DSC is loaded into the I/O Buffer Latch the DMA request, DRQOUT, will be deactivated (see Figure 2 above). The Host Status (HSTAT) SFR Data Stream Command bit is set and the INTRQ interrupt output goes active, if enabled. The Host responds to the interrupt as described above.



Symbol	Description	Var. Osc.	@16 MHz
TCC	Command Cycle Time	6 * Tclcl	375 ns min
TRV	Command Recovery Time	75	75 ns min

Figure 6. UPI-452 Command Cycle Timing

Once INTRQ is deactivated and the DSC has been read by the Host, the DMA request, DRQOUT, is reasserted by the UPI-452. The DMA request then remains active until the transfer is complete or another DSC is loaded into the I/O Buffer Latch.

An Immediate Command written by the internal CPU during a Host DMA FIFO transfer also causes the Host Status flag and INTRQ to go active if enabled. In this case the Immediate Command would not terminate the DMA transfer unless terminated by the Host. The INTRQ line remains active until the Host reads the Host Status (HSTAT) SFR to determine the source of the interrupt.

The net effect of a Data Stream Command (DSC) on DMA data transfer rates is to add an additional factor to the data transfer rate equation. This added factor is shown in Equation 3. An Immediate Command has the same effect on the data transfer rate if the Immediate Command interrupt is recognized by the Host during a DMA transfer. If the DMA transfer is completed before the Immediate Command interrupt is recognized, the effect on the DMA transfer rate depends on whether the block being transmitted is larger than the FIFO channel size. If the block is larger than the programmed FIFO channel size the transfer rate depends on whether the Immediate Command flag or interrupt is recognized between partial block transfers.

The FIFO configuration shown in Equation 3 is arbitrary since there is no way of predicting the size relative to when a DSC would be loaded into the I/O Buffer Latch. The Host DMA rate shown is for a UPI-452 (Memory Mapped or I/O) to 286 System Memory transfer as described earlier. The equations do not account for the latency of intiating the DMA transfer.

Equation 3. Minimum host FIFO DMA Transfer Rate Including Data Stream Command(s)

Minimum Host/FIFO DMA Transfer Rate w/ DSC
= FIFO size* Host DMA 2 cycle time transfer rate + iAPX 286 interrupt response time (Eq. #1)
= (32 bytes* (200 ns + 200 ns)) + 2.3 μs
= 15.1 μs
= 75.5 bus cycles (@10 MHz iAPX286, 200 ns bus cycle)

UPI-452 INTERNAL DMA PROCESSOR

The two identical internal DMA channels allow high speed data transfers from one UPI-452 writable memory space to another. The following UPI-452 memory spaces can be used with internal DMA channels:

Internal Data Memory (RAM)
External Data Memory (RAM)
Special Function Registers (SFR)

The FIFO can be accessed during internal DMA operations by specifying the FIFO IN (FIN) SFR as the DMA Source Address (SAR) or the FIFO OUT (FOUT) SFR as the Destination Address (DAR). Table 6 lists the four types of internal DMA transfers and their respective timings.

Table 6. UPI-452 Internal DMA Controller Cycle Timings

Source	Destination	Machine Cycles**	@12 MHz	@16 MHz
Internal Data Mem. or SFR	Internal Data Mem. or SFR	1	1 μs	750 ns
Internal Data Mem. or SFR	External Data Mem.	1	1 μs	750 ns
External Data Mem.	Internal Data Mem. or SFR	1	1 μs	750 ns
*External Data Memory	External Data Memory	2	2 μs	1.5 μs

NOTES:
*External Data Memory DMA transfer applies to UPI-452 Local Bus only.
**MSC-51 Machine cycle = 12 clock cycles (TCLCL).

FIFO Data Structure and Internal DMA

The effect of Data Stream Commands and Immediate Commands on the internal DMA transfers is essentially the same as the effect on Host FIFO DMA transfers. Recognition also depends upon the programmed DMA Mode, the interrupts enabled, and their priorities. The net internal effect is the same for each possible internal case. The time required to respond to the Immediate or Data Stream Command is a function of the instruction time required. This must be calculated by the user based on the instruction cycle time given in the MSC-51 Instruction Set description in the Intel Microcontroller Handbook.

It is important to note that the internal DMA processor modes and the internal FIFO logic work together to automatically manage internal DMA transfers as data moves into and out of the FIFO. The two most appropriate internal DMA processor modes for the FIFO are FIFO Demand Mode and FIFO Alternate Cycle Mode. In FIFO Demand Mode, once the correct Slave Control and DMA Mode bits are set, the internal Input FIFO channel DMA transfer occurs whenever the Slave Control Input FIFO Request for Service flag is set. The DMA transfer continues until the flag is cleared or when the Input FIFO Read Pointer SFR (IRPR) equals zero. If data continues to be entered by the Host, the internal DMA continues until an internal interrupt of higher priority, if enabled, interrupts the DMA transfer, the internal DMA byte count reaches zero or until the Input FIFO Read Pointer equals zero. A complete description of interrupts and DMA Modes can be found in the UPI-452 Data Sheet.

DMA Modes

The internal DMA processor has four modes of operation. Each DMA channel is software programmable to operate in either Block Mode or Demand Mode. Demand Mode may be further programmed to operate in Burst or Alternate Cycle Mode. Burst Mode causes the internal processor to halt its execution and dedicate its resources exclusively to the DMA transfer. Alternate Cycle Mode causes DMA cycles and instruction cycles to occur alternately. A detailed description of each DMA Mode can be found in the UPI-452 Data Sheet.

INTERFACE LATENCY

The interface latency is the time required to accommodate all of the overhead associated with an individual data transfer. Data transfer rates between the Host system and UPI-452 FIFO, with a block size less than or equal to the programmed FIFO channel size, are calculated using the Host system DMA rate. (see Host DMA description above). In this case, the entire block could be transferred in one operation. The total latency is the time required to accomplish the block DMA transfer, the interrupt response or poll of the Host Status SFR response time, and the time required to initate the Host DMA processor.

A DMA transfer between the Host and UPI-452 FIFO with a block size greater than the programmed FIFO channel size introduces additional overhead. This additional overhead is from three sources; first, is the time to actually perform the DMA transfer. Second, the overhead of initializing the DMA processor, third, the handshaking between each FIFO block required to transfer the entire data block. This could be time to wait for the FIFO to be emptied and/or the interrupt response time to restart the DMA transfer of the next portion of the block. A fourth component may also be the time required to respond to Underrun and Overrun FIFO Errors.

Table 7 shows six typical FIFO Input/Output channel sizes and the Host DMA transfers times for each. The timings shown reflect a 10 MHz system bus two cycle I/O to Memory DMA transfer rate of 2.5 MBytes/second as shown in Equation 1. The times given would be the same for iAPX 286 I/O block move instructions REP INS and REP OUTS as described earlier.

Table 7. Host DMA FIFO Data Transfer Times

FIFO Size:	32	43	64	85	96	128	bytes
Full or Empty	1/4	1/3	1/2	2/3	3/4	Full or Empty	
Time	12.8	17.2	25.6	34.0	38.4	51.2	μs

Table 8 shows six typical FIFO Input/Output channel sizes and the internal DMA processor data transfers times for each. The timings shown are for a UPI-452 single cycle Burst Mode transfer at 16 MHz or 750 ns per machine cycle in or out of the FIFO channels. The

machine cycle time is that of the MSC-51 CPU; 6 states, 2 XTAL2 clock cycles each or 12 clock cycles per machine cycle. Details on the MSC-51 machine cycle timings and operation may be found in the Intel Microcontroller Handbook.

Table 8. UPI-452 Internal DMA FIFO Data Transfer Times

FIFO Size:	32	43	64	85	96	128	bytes
Full or Empty	1/4	1/3	1/2	2/3	3/4	Full or Empty	
Time	24.0	32.3	48.0	64.6	72.0	96.0	μs

A larger than programmed FIFO channel size data block internal DMA transfer requires internal arbitration. The UPI-452 provides a variety of features which support arbitration between the two internal DMA channels and the FIFO. An example is the internal DMA processor FIFO Demand Mode described above. FIFO Demand Mode DMA transfers occur continuously until the Slave Status Request for Service Flag is deactivated. Demand Mode is especially useful for continuous data transfers requiring immediate attention. FIFO Alternate Cycle Mode provides for interleaving DMA transfers and instruction cycles to achieve a maximum of software flexibility. Both internal DMA channels can be used simultaneously to provide continuous transfer for both Input and Output FIFO channels.

Byte by byte transfers between the FIFO and internal CPU timing is a function of the specific instruction cycle time. Of the 111 MCS-51 instructions, 64 require 12 clock cycles, 45 require 24 clock cycles and 2 require 48 clock cycles. Most instructions involving SFRs are 24 clock cycles except accumulator (for example, MOV direct, A) or logical operations (ANL direct, A). Typical instruction and their timings are shown in Table 9.

Oscillator Period: @ 12 MHz = 83.3 ns
@ 16 MHz = 62.5 ns

Table 9. Typical Instruction Cycle Timings

Instruction	Oscillator Periods	@12 MHz	@16 MHz
MOV direct†, A	12	1 μs	750 ns
MOV direct, direct	24	2 μs	1.5 μs

NOTE:
† Direct = 8-bit internal data locations address. This could be an Internal Data RAM location (0–255) or a SFR [i.e., I/O port, control register, etc.]

Byte by byte FIFO data transfers introduce an additional overhead factor not found in internal DMA operations. This factor is the FIFO block size to be transferred; the number of empty locations in the Output channel, or the number of bytes in the Input FIFO channel. As described above in the FIFO Data Structure section, the block size would have to be determined by reading the channel read and write pointer and calculating the space available. Another alternative uses the FIFO Overrun and Underrun Error flags to manage the transfers by accepting error flags. In either case the instructions needed have a significant impact on the internal FIFO data transfer rate latency equation.

A typical effective internal FIFO channel transfer rate using internal DMA is shown in Equation 4. Equation 5 shows the latency using byte by byte transfers with an arbitrary factor added for internal CPU block size calculation. These two equations contrast the effective transfer rates when using internal DMA versus individual instructions to transfer 128 bytes. The effective transfer rate is approximately four times as fast using DMA versus using individual instructions (96 μs with DMA versus 492 μs non-DMA).

Equation 4. Effective Internal FIFO Transfer Time Using Internal DMA

Effective Internal FIFO Transfer Rate with DMA
= FIFO channel size * Internal DMA Burst Mode Single Cycle DMA Time
= 128 Bytes * 750 ns
= 96 μs

Equation 5. Effective FIFO Transfer Time Using Individual Instructions

Effective Internal FIFO Transfer Rate without DMA
= FIFO channel size * Instruction Cycle Time + Block size calculation Time
= 128 Bytes * (24 oscillator periods @ 16 MHz) + 20 instructions (24 oscillator period each @ 16 MHz)
= 128 * 1.5 μs + 300 μs
= 492 μs

FIFO DMA FREEZE MODE INTERFACE

FIFO DMA Freeze Mode provides a means of locking the Host out of the FIFO Input and Output channels. FIFO DMA Freeze Mode can be invoked for a variety of reasons, for example, to reconfigure the UPI-452 Local Expansion Bus, or change the baud rate on the serial channel. The primary reason the FIFO DMA Freeze Mode is provided is to ensure that the Host does not read from or write to the FIFO while the FIFO interface is being altered. ONLY the internal CPU has the capability of altering the FIFO Special Function Registers, and these SFRs can ONLY be altered during FIFO DMA Freeze Mode. FIFO DMA Freeze Mode inhibits Host access of the FIFO while the internal CPU reconfigures the FIFO.

FIFO DMA Freeze Mode should not be arbitrarily invoked while the UPI-452 is in normal operation. Because the external CPU runs asynchronously to the internal CPU, invoking freeze mode without first properly resolving the FIFO Host interface may have serious consequences. Freeze Mode may be invoked only by the internal CPU.

The internal CPU invokes Freeze Mode by setting bit 3 of the Slave Control SFR (SC3). This automatically forces the Slave and Host Status SFR FIFO DMA Freeze Mode to In Progress (SSTAT SST5 = 0, HSTAT SFR HST1 = 1). INTRQ goes active, if enabled by MODE SFR bit 4, whenever FIFO DMA Freeze Mode is invoked to notify the Host. The Host reads the Host Status SFR to determine the source of the interrupt. INTRQ and the Slave and Host Status FIFO DMA Freeze Mode bits are reset by the Host READ of the Host Status SFR.

During FIFO DMA Freeze Mode the Host has access to the Host Status and Control SFRs. All other Host FIFO interface access is inhibited. Table 10 lists the FIFO DMA Freeze Mode status of all slave bus interface Special Function Registers. The internal DMA processor is disabled during FIFO DMA Freeze Mode and the internal CPU has write access to all of the FIFO control SFRs (Table 11).

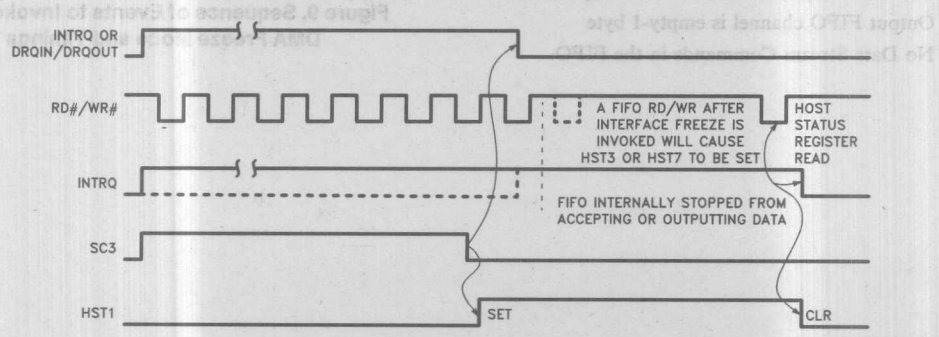
If FIFO DMA Freeze Mode is invoked without stopping the host, only the last two bytes of data written into or read from the FIFO will be valid. The timing diagram for disabling the FIFO module to the external Host interface is illustrated in Figure 7. Due to this synchronization sequence, the UPI-452 might not go into FIFO DMA Freeze Mode immediately after the Slave Control SFR FIFO 7 DMA Freeze Mode bit (SC3) is set = 0. A special bit in the Slave Status SFR (SST5) is provided to indicate the status of the FIFO DMA Freeze Mode. The FIFO DMA Freeze Mode operations described in this section are only valid after SST5 is cleared.

Either the Host or internal CPU can request FIFO DMA Freeze Mode. The first step is to issue an Immediate Command indicating that FIFO DMA Freeze Mode will be invoked. Upon receiving the Immediate Command, the external CPU should complete servicing all pending interrupts and DMA requests, then send an Immediate Command back to the internal CPU acknowledging the FIFO DMA Freeze Mode request. After issuing the first Immediate Command, the internal CPU should not perform any action on the FIFO until FIFO DMA Freeze Mode is invoked. The handshaking is the same in reverse if the HOST CPU initiates FIFO DMA Freeze Mode.

After the slave bus interface is frozen, the internal CPU can perform the operations listed below on the FIFO Special Function Registers. These operations are allowed only during FIFO DMA Freeze Mode. Table 11 summarizes the characteristics of all the FIFO Special Function Registers during Normal and FIFO DMA Freeze Modes.

For FIFO Reconfiguration
1. Changing the Channel Boundary Pointer SFR.
2. Changing the Input and Output Threshold SFR.

To Enhance the testability
3. Writing to the read and write pointers of the Input and Output FIFO's.
4. Writing to and reading the Host Control SFRs.
5. Controlling some bits of Host and Slave Status SFRs.
6. Reading the Immediate Command Out SFR and Writing to the Immediate Command in SFR.



NOTE:
Timing Diagram of disabling of FIFO Module—External Host Interface.

Figure 7. Disabling FIFO to Host Slave Interface Timing Diagram

The sequence of events for invoking FIFO DMA Freeze Mode are listed in Figure 8.

1. Immediate Command to request FIFO DMA Freeze Mode (interrupt)
2. Host/internal CPU interrupt response/service
3. Host/internal CPU clear/service all pending interrupts and FIFO data
4. Internal CPU sets Slave Control (SLCON) FIFO DMA

   Freeze Mode bit = 0, FIFO DMA Freeze Mode, Host Status SFR FIFO DMA Freeze Mode Status bit = 1, INTRQ active (high)
5. Host $\overline{\text{READ}}$ Host Status SFR
6. Internal CPU reconfigures FIFO SFRs
7. Internal CPU resets Slave Control (SLCON) FIFO DMA

   Freeze Mode bit = 1, Normal Mode, Host Status FIFO DMA Freeze Mode Status bit = 0.
8. Internal CPU issues Immediate Command to Host indicating that FIFO DMA Freeze Mode is complete

   or

   Host polls Host Status SFR FIFO DMA Freeze Mode bit to determine end of reconfiguration

Figure 8. Sequence of Events to Invoke FIFO DMA Freeze Mode

EXAMPLE CONFIGURATION

An example of the time required to reconfigure the FIFO 180 degrees, for example from 128 bytes Input to 128 bytes Output, is shown in Figure 9. The example approximates the time based on several assumptions;

1. The FIFO Input channel is full-128 bytes of data
2. Output FIFO channel is empty-1 byte
3. No Data Stream Commands in the FIFO.
4. The Immediate Command interrupt is responded to immediately—highest priority—by Host and internal CPU.
5. Respective interrupt response times
   a. Host (Equation 3 above) = approximately 1.6 μs
   b. Internal CPU is 86 oscillator periods or approximately 5.38 μs worst case.

Event	Time
Immediate Command from Host to UPI-452 to request FIFO DMA Freeze Mode (iAPX286 $\overline{\text{WRITE}}$)	0.30 μs
Internal CPU interrupt response/service	5.38 μs
Internal CPU clears FIFO-128 bytes DMA	96.00 μs
Internal CPU sets Slave Control Freeze Mode bit	0.75 μs
Immediate Command to Host-Freeze Mode in progress Host Immediate Command interrupt response	2.3 μs
Internal CPU reconfigures FIFO SFRs	
Channel Boundary Pointer SFR	0.75 μs
Input Threshold SFR	0.75 μs
Output Threshold SFR	0.75 μs
Internal CPU resets Slave Control (SLCON) Freeze Mode bit = 1, Normal Mode, and automatically resets Host Status FIFO DMA Freeze Mode bit	2.3 μs
Internal CPU writes Immediate Command Out	0.75 μs
Host Immediate Command interrupt service	2.3 μs
Total Minimum Time to Reconfigure FIFO	112.33 μs

Figure 9. Sequence of Events to Invoke FIFO DMA Freeze Mode and Timings

Table 10. Slave Bus Interface Status During FIFO DMA Freezer Mode

Interface Pins;							Operation In Normal Mode	Status In Freeze Mode
$\overline{DACK}$	$\overline{CS}$	A2	A1	A0	$\overline{READ}$	$\overline{WRITE}$		
1	0	0	1	0	0	1	Read Host Status SFR	Operational
1	0	0	1	1	0	1	Read Host Control SFR	Operational
1	0	0	1	1	1	0	Write Host Control SFR	Disabled
1	0	0	0	0	0	1	Data or DMA data from Output Channel	Disabled
1	0	0	0	0	1	0	Data or DMA data to Input Channel	Disabled
1	0	0	0	1	0	1	Data Stream Command from Output Channel	Disabled
1	0	0	0	1	1	0	Data Stream Command to Input Channel	Disabled
1	0	1	0	0	0	1	Read Immediate Command Out from Output Channel	Disabled
1	0	1	0	0	1	0	Write Immediate Command In to Input Channel	Disabled
0	X	X	X	X	0	1	DMA Data from Output Channel	Disabled
0	X	X	X	X	1	0	DMA Data to Input Channel	Disabled

NOTE:
X = don't care

Table 11. FIFO SFR's Characteristics During FIFO DMA Freeze Mode

Label	Name	Normal Operation (SST5 = 1)	Freeze Mode Operation (SST5 = 0)
HCON	Host Control	Not Accessible	Read & Write
HSTAT	Host Status	Read Only	Read & Write
SLCON	Slave Control	Read & Write	Read & Write
SSTAT	Slave Status	Read Only	Read & Write
IEP	Interrupt Enable & Priority	Read & Write	Read & Write
MODE	Mode Register	Read & Write	Read & Write
IWPR	Input FIFO Write Pointer	Read Only	Read & Write
IRPR	Input FIFO Read Pointer	Read Only	Read & Write
OWPR	Output FIFO Write Pointer	Read Only	Read & Write
ORPR	Output FIFO Read Pointer	Read Only	Read & Write
CBP	Channel Boundary Pointer	Read Only	Read & Write
IMIN	Immediate Command In	Read Only	Read & Write
IMONT	Immediate Command Out	Read & Write	Read & Write
FIN	FIFO IN	Read Only	Read Only
CIN	COMMAND IN	Read Only	Read Only
FOUT	FIFO OUT	Read & Write	Read & Write
COUT	COMMAND OUT	Read & Write	Read & Write
ITHR	Input FIFO Threshold	Read Only	Read & Write
OTHR	Other FIFO Threshold	Read Only	Read & Write

ICE™-42
8042 IN-CIRCUIT EMULATOR

- **Precise, Full-Speed, Real-Time Emulation**
  - **— Load, Drive, Timing Characteristics**
  - **— Full-Speed Program RAM**
  - **— Parallel Ports**
  - **— Data Bus**
- **User-Specified Breakpoints**
- **Execution Trace**
  - **— User-Specified Qualifier Registers**
  - **— Conditional Trigger**
  - **— Symbolic Groupings and Display**
  - **— Instruction and Frame Modes**
- **Emulation Timer**
- **Full Symbolic Debugging**
- **Single-Line Assembly and Disassembly for Program Instruction Changes**
- **Macro Commands and Conditional Block Constructs for Automated Debugging Sessions**
- **HELP Facility: ICE™-42 Command Syntax Reference at the Console**
- **User Confidence Test of ICE-42 Hardware**

The ICE-42 module resides in the Intellec Microcomputer Development System and interfaces to any user-designed 8042 or 8041A system through a cable terminating in an 8042 emulator microprocessor and a pin-compatible plug. The emulator processor, together with 2K bytes of user program RAM located in the ICE-42 buffer box, replaces the 8042 device in the user system while maintaining the 8042 electrical and timing characteristics. Powerful Intellec debugging functions are thus extended into the user system. Using the ICE-42 module, the designer can emulate the system's 8042 chip in real-time or single-step mode. Breakpoints allow the user to stop emulation on user-specified conditions, and a trace qualifier feature allows the conditional collection of 1000 frames of trace data. Using the single-line 8042 assembler the user may alter program memory using the 8042 assembler mnemonics and symbolic references, without leaving the emulator environment. Frequently used command sequences can be combined into compound commands and identified as macros with user-defined names.



210818–1

November 1986
Order Number: 210818-002

FUNCTIONAL DESCRIPTION

Integrated Hardware and Software Development

The ICE-42 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-42 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed. Figure 1 shows the ICE-42 emulator connected to a user prototype.

The ICE-42 emulator assists four stages of development.

SOFTWARE DEBUGGING

This emulator operates without being connected to the user's system before any of the user's hardware is available. In this stage ICE-42 debugging capabilities can be used in conjunction with the Intellec text editor and 8042 macro-assembler to facilitate program development.

HARDWARE DEVELOPMENT

The ICE-42 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware.

SYSTEM INTEGRATION

Integration of software and hardware begins when any functional element of the user system hardware is connected to the 8042 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is "system" tested in real-time operation as it becomes available.

SYSTEM TEST

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-42 module is then used for real-time emulation of the 8042 chip to debug the system as a completed unit.

The final product verification test may be performed using the 8742 EPROM version of the 8042 microcomputer. Thus, the ICE-42 module provides the ability to debug a prototype or production system at any stage in its development without introducing extraneous hardware or software test tools.

Symbolic Debugging

The ICE-42 emulator permits the user to define and to use symbolic, rather than absolute, references to program and data memory addresses. Thus, there is no need to recall or look up the addresses of key locations in the program, or to become involved with machine code.

When a symbol is used for memory reference in an ICE-42 emulator command, the emulator supplies the corresponding location as stored in the ICE-42 emulator symbol table. This table can be loaded with the symbol table produced by the assembler during application program assembly. The user obtains the symbol table during software preparation simply by using the "DEBUG" switch in the 8042 macroassembler. Furthermore, the user interactively modifies the emulator symbol table by adding new symbols or changing or deleting old ones. This feature provides great flexibility in debugging and minimizes the need to work with hexadecimal values.

Through symbolic references in combination with other features of the emulator, the user can easily:

- Interpret the results of emulation activity collected during trace.
- Disassemble program memory to mnemonics, or assemble mnemonic instructions to executable code.
- Reference labels or addresses defined in a user program.

Automated Debugging and Testing

MACRO COMMAND

A macro is a set of commands given a name. A group of commands executed frequently can be defined as a macro. The user executes the group of commands by typing a colon followed by the macro name. Up to ten parameters may be passed to the macro.

Macro commands can be defined at the beginning of a debug session and then used throughout the whole session. One or more macro definitions can be saved on diskette for later use. The Intellec text editor may be used to edit the macro file. The macro definitions are easy to include in any later emulation session.

The power of the development system can be applied to manufacturing testing as well as development by writing test sequences as macros. The macros are stored on diskettes for use during system test.

COMPOUND COMMAND

Compound commands provide conditional execution of commands (IF command) and execution of commands repeatedly until certain conditions are met (COUNT, REPEAT commands).

Compound commands may be nested any number of times, and may be used in macro commands.

Example:

```
*DEFINE .I=0          ;Define symbol .I to 0
*COUNT 100H           ;Repeat the following
                       commands 100H times.
.*IF .I AND 1 THEN    ;Check if .I is odd
..*CBYTE.I=.I         ;Fill the memory at
                       location .I to value .I
..*END
.*.I=.I+1             ;Increment .I by 1.
.*END                 ;Command executes
                       upon carriage-return
                       after END
```

(The asterisks are system prompts; the dots indicate the nesting level of compound commands.)

Operating Modes

The ICE-42 software is an Intellec RAM-based program that provides easy-to-use commands for initiating emulation, defining breakpoints, controlling trace data collection, and displaying and controlling system parameters. ICE-42 commands are configured with a broad range of modifiers that provide maximum flexibility in describing the operation to be performed.

EMULATION

The ICE-42 module can emulate the operation of prototype 8042 system, at real-time speed (up to 12 MHz) or in single steps. Emulation commands to the ICE-42 module control the process of setting up, running, and halting an emulation of the user's 8042-based system. Breakpoints and tracepoints enable the ICE-42 emulator to halt emulation and provide a detailed trace of execution in any part of the user's program. A summary of the emulation commands is shown in Table 1.

Table 1. Major Emulation Commands

Command	Description
GO	Begins real-time emulation and optionally specifies break conditions.
BR0, BR1, BR	Sets or displays either or both Breakpoint Registers used for stopping real-time emulation.
STEP	Performs single-step emulation.
QR0, QR1	Specifies match conditions for qualified trace.
TR	Specifies or displays trace-data collection conditions and optionally sets Qualifier Register (QR0, QR1).
Synchronization Line Commands	Sets and displays status of synchronization line outputs or latched inputs. Used to allow real-time emulation or trace to start and stop synchronously with external events.

Breakpoints

The ICE-42 hardware includes two breakpoint registers that allow halting of emulation when specified conditions are met. The emulator continuously compares the values stored in the breakpoint registers with the status of specified address, opcode, operand, or port values, and halts emulation when this comparison is satisfied. When an instruction initiates a break, that instruction is executed completely before the break takes place. The ICE-42 emulator then regains control of the console and enters the interrogation mode. With the breakpoint feature, the user can request an emulation break when the program:

- Executes an instruction at a specific address or within a range of addresses.
- Executes a particular opcode.
- Receives a specific signal on a port pin.
- Fetches a particular operand from the user program memory
- Fetches an operand from a specific address in program memory.

Tracing is used with real-time and single-step emulation to record diagnostic information in the trace buffer as a program is executed. The information collected includes opcodes executed, port values, and memory addresses. The ICE-42 emulator collects 1000 frames of trace data.

If desired this information can be displayed as assembler instruction mnemonics for analysis during interrogation or single-step mode. The trace-collection facility may be set to run conditionally or unconditionally. Two unique trace qualifier registers, specified in the same way as breakpoint registers, govern to condition trace data collection to take place as follows:

- Under all conditions (forever).
- Only while the trace qualifier is satisfied.
- For the frames or instructions preceding the time when a trace qualifier is first satisfied (pre-trigger trace).
- For the frames or instructions after a trace qualifier is first satisfied (post-triggered trace).

Table 2 shows an example of trace display.

Table 2. Trace Display (Instruction Mode)

FRAME	LOC	OBJ	INSTRUCTION	P1	P2	T0	T1	DBYIN	YOUT	YSTS	TOVF
0000:	100H	2355	MOV A,#55H	FFH	FFH	0	0	66H	DFH	02H	0
0004:	102H	39	OUTL P1,A	FFH	FFH	0	0	66H	DFH	02H	0
0008:	103H	3A	OUTL P2,A	55H	FFH	0	0	66H	DFH	02H	0
0012:	104H	22	IN A,DBB	55H	55H	0	0	66H		02H	0
0014:	105H	37	CPL A	55H	55H	0	0		DFH	02H	0
0016:	106H	02	OUT DBB,A	55H	55H	0	0	66H		00H	0
0018:	107H	BA03	MOV R2,#03H	55H	55H	0	0	66H	99H	00H	0
0022:	109H	B840	MOV R0,#.TABLE0	55H	55H	0	0	66H	99H	01H	0
0026:	10BH	B960	MOV R1,#.TABLE1	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0030:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0032:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0034:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0036:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0038:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0042:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0044:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0046:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0048:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0050:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0054:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0056:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0058:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0060:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0062:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
0066:	113H	00	NOP	55H	55H	0	0		99H	01H	0

210818-2

INTERROGATION AND UTILITY

Interrogation and utility commands give convenient access to detailed information about the user program and the state of the 8042 that is useful in debugging hardware and software. Changes can be made in memory and in the 8042 registers, flags, and port values. Commands are also provided for various utility operations such as loading and saving program files, defining symbols, displaying trace data, controlling system synchronization and returning control to ISIS-II. A summary of the basic interrogation and utility commands is shown in Table 3. Two additional time-saving emulator features are discussed below.

Single-Line Assembler/Disassembler

The single-line assembler/disassembler (ASM and DASM commands) permits the designer to examine and alter program memory using assembly language mnemonics, without leaving the emulator environment or requiring time-consuming program reassembly. When assembling new mnemonic instructions into program memory, previously defined symbolic references (from the original program assembly, or subsequently defined during the emulation session)

Table 3. Major Interrogation and Utility Commands

Command	Description
HELP	Displays help messages for ICE-42 emulator command-entry assistance.
LOAD	Loads user object program (8042 code) into user-program memory, and user symbols into ICE-42 emulator symbol table.
SAVE	Saves ICE-42 emulator symbol table and/or user object program in ISIS-II hexadecimal file.
LIST	Copies all emulator console input and output to ISIS-II file.
EXIT	Terminates ICE-42 emulator operation.
DEFINE	Defines ICE-42 emulator symbol or macro.
REMOVE	Removes ICE-42 emulator symbol or macro.
ASM	Assembles mnemonic instructions into user-program memory.
DASM	Disassembles and displays user-program memory contents.
Change/Display Commands	Change or display value of symbolic reference in ICE-42 emulator symbol table, contents of key-word references (including registers, I/O ports, and status flags), or memory references.
EVALUATE	Evaluates expression and displays resulting value.
MACRO	Displays ICE-42 macro or macros.
INTERRUPT	Displays contents for the Data Bus and timer interrupt registers.
SECONDS	Displays contents of emulation timer, in microseconds.
Trace Commands	Position trace buffer pointer and select format for trace display.
PRINT	Displays trace data pointed to by trace buffer pointer.
MODE	Sets or displays the emulation mode, 8041A or 8042.

Table 4. HELP Command

```
*HELP
Help is available for the following items. Type HELP followed by the item name.
The help items cannot be abbreviated. (For more information, type HELP HELP or
HELP INFO.)
Emulation:   Trace Collection:          Misc:        <address>
GO GR SYO    TR   QR   QR0  QR1  SY1   BASE         <CPU*keyword>
BR BR0BR1                              DISABLE      <expr>
STEP         Trace Display:            ENABLE       <ICE42 #keyword>
             TRACE  MOVE    PRINT      ERROR        <identifier>
             OLDEST NEWEST             EVALUATE     <instruction>
                                       HELP         <masked#constant>
Change/       Display/ Define/ Remove: INFO         <match*cond>
<CHANGE>      REMOVE   CBYTE           <LIGHTS>     <numeric*constant>
<DISPLAY>     SYMBOL   DBYTE   DASM    LIST         <partition>
REGISTER      RESET            ASM     LOAD         <string>
                                       MODE
SECONDS       WRITE                    SAVE         <string*constant>
DEFINE        STACK            SY      SUFFIX       <symbolic*ref>
                                       SYMBOLIC     <mode>
Macro:             Compound                         <trace*reference>
DEFINE       DIR   Commands:                        <unlimited*match*cond>
DISABLE      ENABLE COUNT                           <user*symbols>
INCLUDE      PUT    IF
<MACRO*DISPLAY>      REPEAT
<MACRO*INVOCATION>
*
*
*HELP IF
IF - The conditional command allows conditional execution of one or more commands
based on the values of boolean conditions.
      IF <expr> 'THEN  <cr>              <true*list>::='<command> <cr> @
        <true*list>                      <false*list>::='<command> <cr> @
        'ORIF <expr> <cr>                <command>::=An ICE-42 command.
        <true*list> @
        'ELSE <cr>
        <false*list>
        END
The <expr>s are evaluated in order as 16-bit unsigned integers. If one is
reached whose value has low-order bit 1 (TRUE), all commands in the <true*list>
following that <expr> are then executed and all commands in the other <true*-
list>s and in the <false*list> are skipped. If all <expr>s have value with low-
order bit 0 (FALSE), then all commands in all <true*list>s are skipped and, if
ELSE is present, all commands in the <false*list> are executed.
     (EX:  IF .LOOP=5 THEN
             STEP
             ELSE
             GO
             END)
*
*
*
*
*EXIT
```

210818-3

may be used in the instruction operand field. The emulator supplies the absolute address or data values as stored in the emulator symbol table. These features eliminate user time spent translating to and from machine code and searching for absolute addresses, with a corresponding reduction in transcription errors.

HELP

The HELP file allows display of ICE-42 command syntax information at the Intellec console. By typing "HELP", a listing of all items for which help messages are available is displayed. Typing "HELP <Item>" then displays relevant information about the item requested, including typical usage examples. Table 4 shows some sample HELP messages.

EMULATION ACCURACY

The speed and interface demands of a high-performance single-chip microcomputer require extremely accurate emulation, including full-speed, real-time operation with the full function of the microcomputer. The ICE-42 module achieves accurate emulation with an 8042 emulator chip, a special configuration of the 8042 microcomputer family, as its emulation processor.

Each of the 40 pins on the user plug is connected directly to the corresponding 8042 pin on the emulator chip. Thus the user system sees the emulator as an 8042 microcomputer at the 8042 socket. The resulting characteristics provide extremely accurate emulation of the 8042 including speed, timing characteristics, load and drive values, and crystal operation. However, the emulator may draw more power from the user system than a standard 8042 family device.

Additional emulator processor pins provide signals such as internal address, data, clock, and control lines to the emulator buffer box. These signals let static RAM in the buffer box substitute for on-chip program ROM or EPROM. The emulator chip also gives the ICE module "back-door" access to internal chip operation, allowing the emulator to break and trace execution without interfering with the values on the user-system pins.

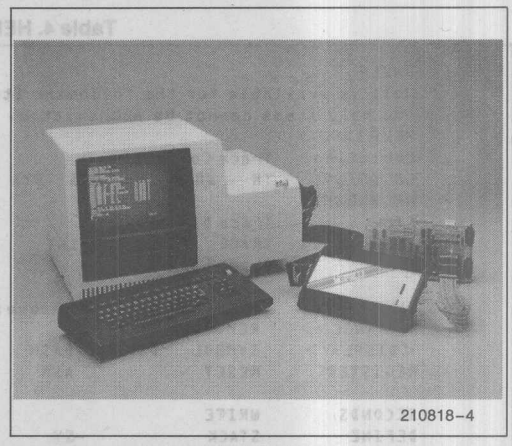


Figure 1. A typical 8042 Development Configuration. The host system is an Intellec Series IV. The ICE-42 module is connected to a user prototype system.

SPECIFICATIONS

ICE™-42 Operating Requirements

Intellec Model 800, Series II, Series III, or Series IV Microcomputer Development System (64K RAM required)

System console (Model 800 only)

Intellec Diskette Operating System: ISIS (Version 3.4 or later).

Equipment Supplied

- Printed circuit boards (2)
- Emulation buffer box, Intellec interface cables, and user-interface cable with 8042 emulation processor
- Crystal power accessory
- Operating instructions manuals
- Diskette-based ICE-42 software (single and double density)

Emulation Clock

User's system clock (up to 12 MHz) or ICE-42 crystal power accessory (12 MHz)

Environmental Characteristics

Operating Temperature: 0°C to 40°C
Operating Humidity: Up to 95% relative humidity without condensation.

Physical Characteristics

Printed Circuit Boards

Width: 12.00 in. (30.48 cm)
Height: 6.75 in. (17.15 cm)
Depth: 0.50 in. (1.27 cm)

Buffer Box

Width: 8.00 in. (20.32 cm)
Length: 12.00 in. (30.48 cm)
Depth: 1.75 in. (4.44 cm)
Weight: 4.0 lb (1.81 kg)

Electrical Characteristics

DC Power Requirements (from Intellec® system)

V_{CC} = +5V, ±5%
I_{CC} = 13.2A max; 11.0A typical
V_{DD} = +12V, ±5%
I_{DD} = 0.1A max; 0.05A typical
V_{BB} = −10V, ±5%
I_{BB} = 0.05A max; 0.01A typical

User plug characteristics at 8042 socket—Same as 8042 or 8742 except that the user system sees an added load of 25 pF capacitance and 50 μA leakage from the ICE-42 emulator user plug at ports 1, 2, T0, and T1.

ORDERING INFORMATION

Part Number	Description
ICE-42	8042 Microcontroller In-Circuit Emulator, cable assembly and interactive diskette software



iUP-200A/iUP-201A UNIVERSAL PROM PROGRAMMERS

MAJOR iUP-200A/iUP-201A FEATURES:

- **Personality Module Plug-Ins Provide Industry First Support for Intel and Intel Compatible EPROMs, EEPROMs, KEPROM, Microcontrollers, and other Programmable Devices.**
- **Powerful PROM Programming Software (iPPS) Makes Programming Easy with Intellec® Development System, iNDS-II Networks, iPDS Personal Development System, IBM P/C, X/T, A/T, and PC DOS Compatibles.**
- **New Modules Provide Industry-Fastest, Intelligent Programming Algorithms to Dramatically Shorten Programming Times.**
- **iUP-200A Provides On-Line Operation with a Built-In Serial RS232 Interface and Software for a Growing List of Environments.**
- **iUP-201A Provides Same On-Line Performance and Adds Keyboard and Display for Stand-Alone Use.**
- **iUP-201A Stand-Alone Capability Includes Device Previewing, Editing, Duplication, and Download from any Source Over RS232C Port.**
- **Regular Updates and Add-Ons Have Maintained Even the Earliest iUP-200 and iUP-201 Users at the State-of-Art.**

The iUP-200A and iUP-201A universal programmers program and verify data in all the Intel and Intel compatible, programmable devices (EPROMs and EEPROMs). They can also program the EPROM memory portions of Intel's single-chip microcomputer and peripheral devices. The iUP-200A and iUP-201A universal programmers provide on-line programming and verification in a growing variety of development environments using the Intel PROM programming software (iPPS). In addition, the iUP-201A universal programmer supports off-line, stand-alone program editing, EPROM duplication, and EPROM memory locking. The iUP-200A universal programmer is expandable to an iUP-201A model.



210319-1

October 1986
Order Number: 210319-004

FUNCTIONAL DESCRIPTION

The iUP-200A universal programmer operates in on-line mode. The iUP-201A universal programmer operates in both on-line and off-line mode.

On-Line System Hardware

The iUP-200A and iUP-201A universal programmers are free-standing units that, when connected to a host computer with at least 64K bytes of memory, provide on-line EPROM programming and verification of Intel programmable devices. In addition, the universal programmer can read the contents of the ROM versions of these devices.

The universal programmer communicates with the host through a standard RS232C serial data link. Different versions of the iUP-200A and iUP-201A are equipped with different cables, including the cable most commonly used for interfacing to that host. Care should be taken that the version with the correct cable for your particular system is selected, as cable requirements can vary with your host configuration. A serial converter is needed when using the MDS 800 as a host system. (Serial converters are available from other manufacturers).

Each universal programmer contains the CPU, selectable power supply, static RAM, programmable timer, interface for personality modules, RS232C interface for the host system, and control firmware in EPROM. The iUP-201A also has a keyboard and display.

A personality module adapts the universal programmer to a family of EPROM devices; it contains all the hardware and firmware necessary to program either a family of Intel EPROMs or a single Intel device. The user inserts the personality module into the universal programmer front panel.

Figure 1 shows the iUP-200A on-line system configurations, and Figure 2 shows the on-line system data flow.

On-Line System Software

The iUP-200A and iUP201A includes your choice of one copy of Intel's PROM Programming software iPPS, selected from a growing list of versions for different operating systems and hosts. Each version includes the software implementation designed for that host and O.S. and the RS232C cable most commonly used. Additional versions may be purchased separately if you decide to change hosts at a later date. The iPPS software provides user control through an easy-to-use interactive interface. The iPPS software performs the following functions to make EPROM programming quick and easy:

- Reads EPROMs and ROMs
- Programs EPROMs directly or from a file
- Verifies EPROM data with buffer data
- Locks EPROM memory from unauthorized access (on devices which support this feature)
- Prints EPROM contents on the network or development system printer
- Performs interactive formatting operations such as interleaving, nibble swapping, bit reversal, and block moves
- Programs multiple EPROMs from the source file, prompting the user to insert new EPROMs
- Uses a buffer to change EPROM contents

All iPPS commands, as well as program address and data information, are entered through the host system ASCII keyboard and displayed on the system CRT. Table 1 summarizes the iPPS commands.

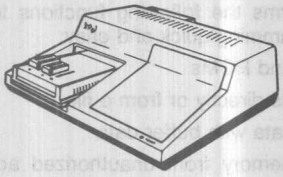
The iPPS software lets the user load programs into an EPROM from host system memory or directly from a disk file. Access to the disk lets the user create and manipulate data in a virtual buffer with an address range up to 16M. This large block of data can be formatted into different EPROM word sizes for program storage into several different EPROM types. In addition, a program stored in the target EPROM, the host system memory, or a system disk file can be interleaved with a second program and entered into a specific target EPROM or EPROMs.

The iPPS software supports data manipulation in any Intel format: 8080 hexadecimal ASCII, 8080 absolute object, 8086 hexadecimal ASCII, 8086 absolute object, and 80286 absolute object. Addresses and data can be displayed in binary, octal, decimal, or hexadecimal. The user can easily change default data formats as well as number bases.

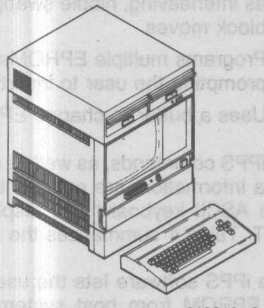
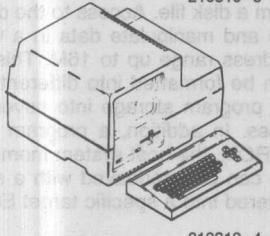
The user invokes the iPPS software from the operating system. Intellec and iPDS Development Systems running ISIS allow running the software under control of ISIS submit files freeing the operator from repetitious command entry.

System Expansion

The iUP-200A universal programmer can be easily upgraded (by the user) to an iUP-201A universal programmer for off-line operation. The upgrade kit (iUP-PAK-A) is available from Intel or your local Intel distributor.



210319-2

MODEL NUMBER	HOST SUPPORTED	S/W OPERATING SYSTEM ENVIRONMENT	RS232C CABLE INCLUDED*	I/O PORT USED
iUP200A211A OR iUP200A212B	 210319-3	ISIS II	INTELLEC-STYLE	CH1 OR 2
iUP200A213C	 210319-4	iNDX	INTELLEC-STYLE	CH1 OR 2
iUP200A216D	 210319-5	PC-DOS	PC OR XT STYLE	COM 1 OR 2
iUP200A217D	 210319-6	PC-DOC	PC AT STYLE	COM 1 OR 2

*RS232C CABLES: INTELLEC STYLE—DB 25 PIN MALE TO DB 25 PIN MALE, NULL MODEM CABLE.
PC XT STYLE — DB 25 PIN FEMALE TO DB 25 PIN MALE, NULL MODEM CABLE.
PC AT STYLE — DB 9 PIN FEMALE TO DB 25 PIN MALE CABLE.

Figure 1. On-Line System Configurations

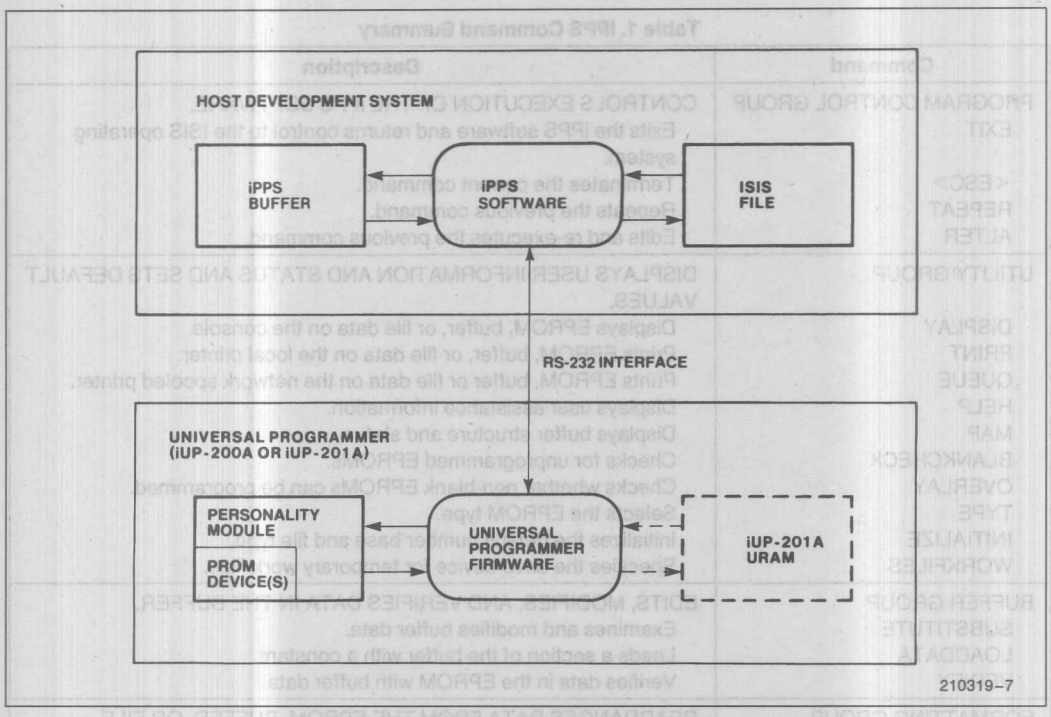


Figure 2. On-Line System Data Flow

Off-Line System

The iUP-201A universal programmer has all the on-line features of the iUP-200A universal programmer plus off-line editing, EPROM duplication, program verification, and locking of EPROM memory independent of the host system. The iUP-201A universal programmer also accepts Intel hexadecimal programs developed on non-Intel development systems. Just a few keystrokes download the program into the iUP RAM for editing and loading into a EPROM.

Off-line commands are entered using the off-line command keys summarized in Table 2.

In addition to the hardware components included as part of the iUP-200A, the iUP-201A contains a 24-character alphanumeric display, full hexadecimal 12-function keypad, and 32K bytes of iUP RAM. Figure 3 illustrates the iUP-201A keyboard and display.

The two logical devices accessible during off-line operation are the EPROM device and the iUP RAM. A typical operation is copying the data from an EPROM (or ROM) into the iUP RAM, modifying this data in iUP RAM, and programming the modified data back into a EPROM device. The address range of the iUP RAM is automatically determined by the universal programmer when EPROM type selection is made. Figure 4 shows the off-line system data flow.

SYSTEM DIAGNOSTICS

Both the iUP-200A and iUP-201A universal programmers include self-contained system diagnostics that verify system operation and aid the user in fault isolation. Diagnostics are performed on the power supply, CPU internal firmware ROM, internal RAM, timer, the iUP-201A keyboard, and the iUP RAM. In addition, tests are made on any personality module installed in the programmer the first time the module is accessed. The personality module tests include the power select circuitry and module firmware. Straightforward messages are provided on the development system display in on-line mode and on the iUP-201A display in off-line mode.

PERSONALITY MODULES

A personality module is the interface between the iUP-200A/iUP-201A universal programmer (or an iPDS system) and a selected EPROM (or ROM). Personality modules contain all the hardware and

Table 1. iPPS Command Summary

Command	Description
PROGRAM CONTROL GROUP	CONTROLS EXECUTION OF THE iPPS SOFTWARE.
EXIT	Exits the iPPS software and returns control to the ISIS operating system.
<ESC>	Terminates the current command.
REPEAT	Repeats the previous command.
ALTER	Edits and re-executes the previous command.
UTILITY GROUP	DISPLAYS USER INFORMATION AND STATUS AND SETS DEFAULT VALUES.
DISPLAY	Displays EPROM, buffer, or file data on the console.
PRINT	Prints EPROM, buffer, or file data on the local printer.
QUEUE	Prints EPROM, buffer or file data on the network spooled printer.
HELP	Displays user assistance information.
MAP	Displays buffer structure and status.
BLANKCHECK	Checks for unprogrammed EPROMs.
OVERLAY	Checks whether non-blank EPROMs can be programmed.
TYPE	Selects the EPROM type.
INITIALIZE	Initializes the default number base and file type.
WORKFILES	Specifies the drive device for temporary work files.
BUFFER GROUP	EDITS, MODIFIES, AND VERIFIES DATA IN THE BUFFER.
SUBSTITUTE	Examines and modifies buffer data.
LOADDATA	Loads a section of the buffer with a constant.
VERIFY	Verifies data in the EPROM with buffer data.
FORMATTING GROUP	REARRANGES DATA FROM THE EPROM, BUFFER, OR FILE.
FORMAT	Formats and interleaves buffer, EPROM, or file data.
COPY GROUP	COPIES DATA FROM ONE DEVICE TO ANOTHER.
COPY (file to PROM)	Programs the EPROM with data in a file on disk.
COPY (PROM to file)	Saves EPROM data in a file on disk.
COPY (buffer to PROM)	Programs the EPROM with data from the buffer.
COPY (PROM to buffer)	Loads the buffer with data in the EPROM.
COPY (buffer to file)	Saves the contents of the buffer in a file on disk.
COPY (file to buffer)	Loads the buffer from a file on disk.
COPY (file to URAM)	Loads file data into the iUP RAM (iUP-201A model only).
COPY (URAM to file)	Saves iUP URAM data in a file on disk (iUP-201A model only).
COPY (buffer to URAM)	Loads the buffer into the iUP URAM (iUP-201A model only).
COPY (URAM to buffer)	Loads iUP URAM data into the buffer (iUP-201A model only).
SECURITY GROUP	LOCKS SELECTED DEVICES TO PREVENT UNAUTHORIZED ACCESS.
KEYLOCK	Locks the EPROM from unauthorized access.

firmware for reading and programming a family of Intel devices. Each personality module is a single molded unit inserted into the front panel of the universal programmer. A wide variety of personality modules and adaptors are available for Intel programmable devices. New modules and adaptors allow you to keep abreast of the newest Intel devices, programming algorithms, and device packages while protecting your equipment investment. Refer to the data sheet on "PROM Programming Personality Modules" for a complete list of available support.

Each personality module connects to the universal programmer through a 41-pin connector. Module firmware is uploaded into the iUP RAM and executed by the internal processor. The personality module firmware contains routines necessary to read and program a family of EPROMs. In addition, the personality module sends specific information about the selected EPROM to the universal programmer to help perform EPROM device integrity checks.

Table 2. Off-Line Command Keys Summary

Key	Function
ON LINE	Selects either on-line or off-line operation. When on-line, all other function keys are disabled.
DEVICE SELECT	Selects the EPROM type when using a personality module able to program multiple EPROM devices.
VER	Verifies the contents of the installed EPROM device with the contents of the iUP RAM. The universal programmer display indicates the address and the XOR of any mismatches.
PROG	Performs a device blank check and then programs the target EPROM with data from the iUP RAM. If the blank check fails, pressing PROG again peforms an overlay check to verify that non-blank EPROMs can be programmed.
ROM TO RAM	Loads the iUP RAM with the data from the EPROM device installed in the personality module.
CLEAR	Terminates the current off-line function, clears a user entry, or restores the display after an error.
ENTER	Transfers information from the universal programmer display (addresses, data, or baud rate) into the iUP RAM.
SHIFT + ADDR 0	Selects an address field for display.
SHIFT + DATA 1	Selects a data field for keypad editing and entry.
SHIFT + FILL 2	Loads a contiguous section of iUP RAM locations with a constant.
SHIFT + LOAD 3	Downloads Intel hexadecimal data from any development system which has an RS232C port.
SHIFT + LOCK 4	Locks a EPROM from unauthorized access.

210319-8

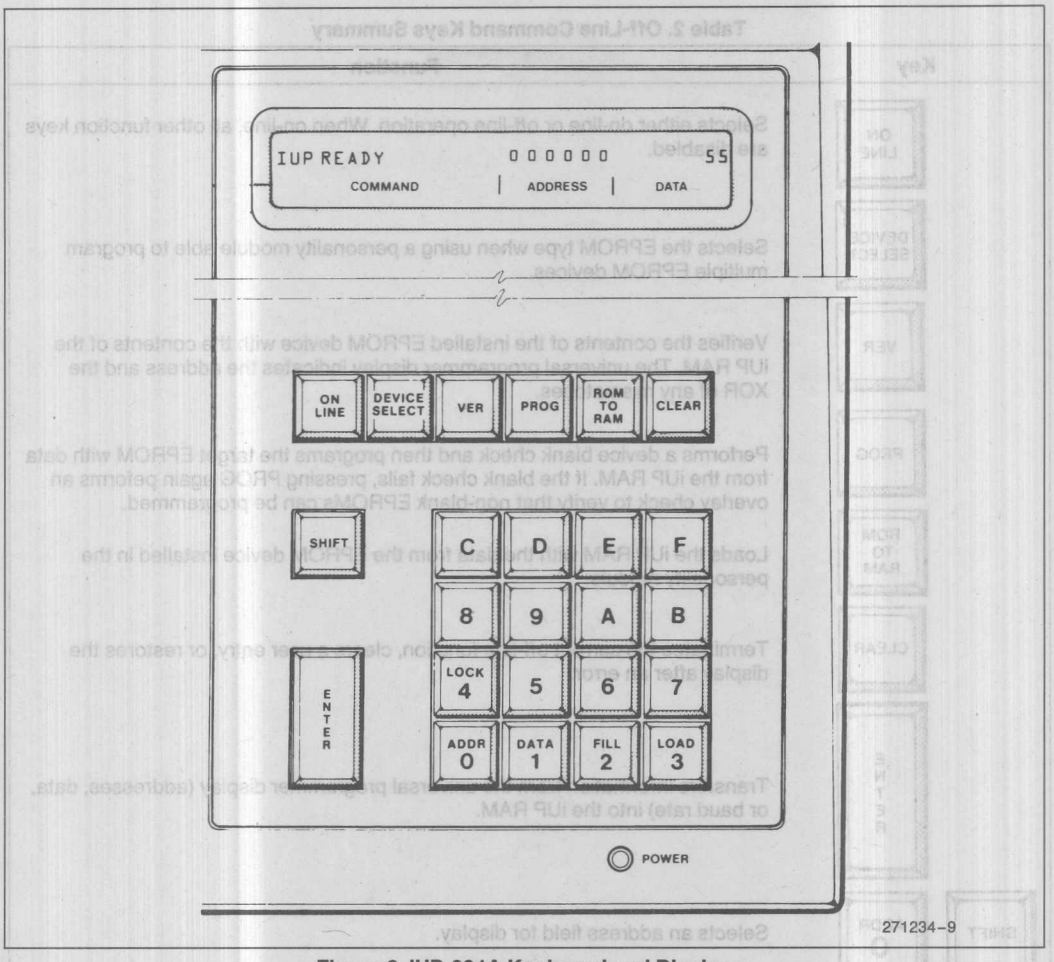


Figure 3. iUP-201A Keyboard and Display

LEDs on each personality module indicate operational status. On some personality modules a column of LEDs indicate which EPROM device type the user has selected. On some personality modules an LED below the socket indicates which socket is to be used. A red indicator light tells the user when power is being supplied to the selected device. Figure 5 shows a selection fo some of the personality modules supported on the universal programmer.

In addition to the testing done by the iUP system self-tests, each personality module contains diagnostic firmware that performs selected EPROM tests and indicates status. These tests are performed in both on-line and off-line modes. The EPROM installation test verifies that the device is installed in the module correctly and that the ZIF socket is closed. The EPROM blank check determines whether a device is blank. The universal programmer automatically determines whether the blank state is all zeros or all ones. The overlay check (performed when a EPROM is not blank) determines which bits are programmed, compares those bits against the program to be loaded, and allows programming to continue if they match. As with the system self-tests, straight-forward messages are provided. The user can invoke all of the EPROM device integrity checks except the installation test (which occurs automatically any time an operation is selected).

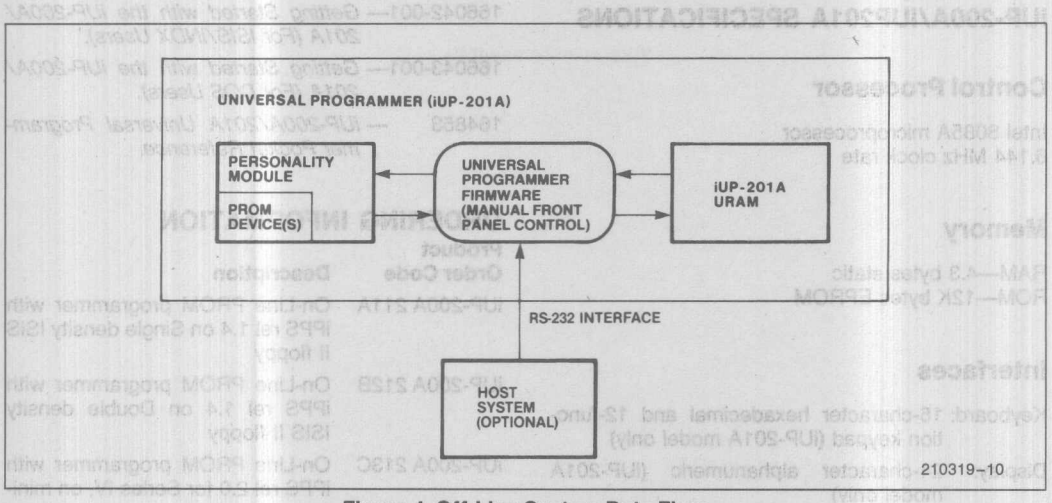


Figure 4. Off-Line System Data Flow



Figure 5. Personality Modules

iUP-200A/iUP201A SPECIFICATIONS

Control Processor

Intel 8085A microprocessor
6.144 MHz clock rate

Memory

RAM—4.3 bytes static
ROM—12K bytes EPROM

Interfaces

Keyboard: 16-character hexadecimal and 12-function keypad (iUP-201A model only)

Display: 24-character alphanumeric (iUP-201A model only)

Software

Monitor— system controller in pre-programmed EPROM

iPPS — Intel PROM programming software on supplied diskette

Physical Characteristics

Depth: 15 inches (38.1 cm)
Width: 15 inches (38.1 cm)
Height: 6 inches (15.2 cm)
Weight: 15 pounds (6.9 kg)

Electrical Characteristics

Selectable 100, 120, 200, or 240 Vac ± 10%; 50-60 Hz

Maximum power consumption—80 watts

Environmental Characteristics

Reading Temperature:	10°C to 40°C
Programming Temperature:	25°C ±5°
Operating Humidity:	10% to 85% relative humidity

Reference Material

166041-001— *iUP-200A/201A Universal Programmer User's Guide.*

166042-001— *Getting Started with the iUP-200A/201A (For ISIS/iNDX Users).*

166043-001— *Getting Started with the iUP-200A/201A (For DOS Users).*

164853 — *iUP-200A/201A Universal Programmer Pocket Reference.*

ORDERING INFORMATION

Product Order Code	Description
iUP-200A 211A	On-Line PROM programmer with iPPS rel 1.4 on Single density ISIS II floppy
iUP-200A 212B	On-Line PROM programmer with iPPS rel 1.4 on Double density ISIS II floppy
iUP-200A 213C	On-Line PROM programmer with iPPS rel 2.0 for Series IV, on mini-floppy
iUP-200A 216D	On-Line PROM programmer with iPPS rel 2.0 for PC/DOS, and cable for PC or XT
iUP-200A 217D	On-Line PROM programmer with iPPS rel 2.0 for PC/DOS, and cable for AT
iUP-201A 211A	Off-Line and on-line PROM programmer with iPPS rel 1.4 on Single density ISIS II floppy
iUP-201A 212B	Off-Line and on-line PROM programmer with iPPS rel 1.4 on Double density ISIS II floppy
iUP-201A 213C	Off-Line and on-line PROM programmer with iPPS rel 2.0 for Series IV on mini-floppy
iUP-201A 216D	Off-Line and on-line PROM programmer with iPPS rel 2.0 for PC/DOS, and cable for PC or XT
iUP-201A 217D	Off-Line and on-line PROM programmer with iPPS rel 2.0 for PC/DOS, and cable for AT
iUP-200/201 U1* Upgrade Kit	Upgrades an iUP-200/201 universal programmer to an iUP-200A/201A universal programmer
iUP-PAK-A Upgrade Kit	Upgrades an iUP-200/A universal programmer to an iUP-201A universal programmer

*Most personality modules can be used only with an iUP-200A/201A universal programmer or an iUP-200/iUP201 universal programmer upgraded to an A with the iUP-200/iUP-201 U1 upgrade kit. If used in an iPDS, most personality modules require version 1.4 of the iPPS software.

Product Order Code	Description
iUP-PAK-A Upgrade Kit	Upgrades an iUP-200A universal programmer to an iUP-201A universal programmer

Software Sold Separately

Product Order Code	Description
211A	PROM programming software rel 1.4 on Single density ISIS II floppy
212B	PROM programming software rel 1.4 on Double density ISIS II floppy
213C	PROM programming software rel 2.0 for Series IV on mini-floppy
216D	PROM programming software rel 2.0 for PC/DOS with cable for PC or PC XT
217D	PROM programming software rel 2.0 for PC/DOS with cable for PC AT
219F	PROM programming software for iPDS on mini-floppy

Product Order Code	Description
iUP-PAK-A Upgrade Kit	Upgrades an iUP-200A universal programmer to an iUP-201A universal programmer

Software Sold Separately

Product Order Code	Description
211A	PROM programming software rel 1.4 on Single density ISIS II floppy
212D	PROM programming software rel 1.4 on Double density ISIS II floppy

Product Order Code	Description
213C	PROM programming software rel 2.0 for Series IV on mini-floppy
216D	PROM programming software rel 2.0 for PC\DOS with cable for PC or PC XT
217D	PROM programming software rel 2.0 for PC\DOS with cable for PC AT
219F	PROM programming software for iPDS on mini-floppy

Alphanumeric Terminal Controllers

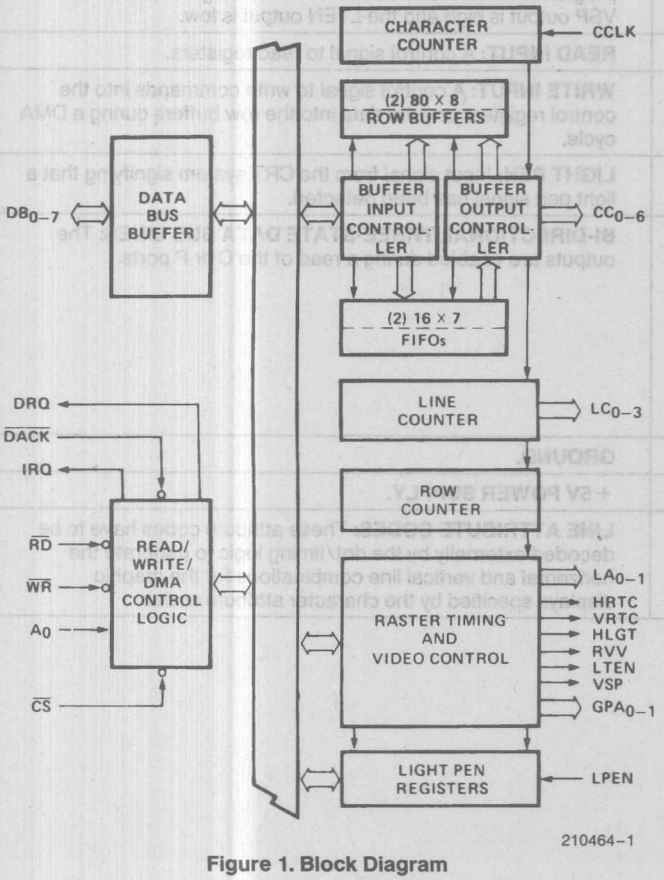
6

6 Alphanumeric Terminal Controllers

8275H
PROGRAMMABLE CRT CONTROLLER

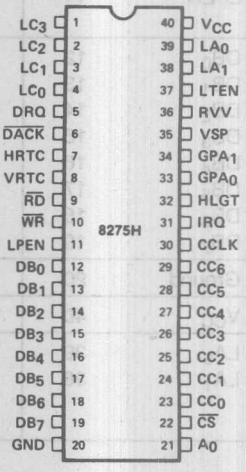
- Programmable Screen and Character Format
- 6 Independent Visual Field Attributes
- 11 Visual Character Attributes (Graphic Capability)
- Cursor Control (4 Types)
- Light Pen Detection and Registers
- MCS-51®, MCS-85®, iAPX 86, and iAPX 88 Compatible
- Dual Row Buffers
- Programmable DMA Burst Mode
- Single +5V Supply
- High Performance HMOS-II

The Intel® 8275H Programmable CRT Controller is a single chip device to interface CRT raster scan displays with Intel® microcomputer systems. It is manufactured on Intel's advanced HMOS-II process. Its primary function is to refresh the display by buffering the information from main memory and keeping track of the display position of the screen. The flexibility designed in the 8275H will allow simple interface to almost any raster scan CRT display with a minimum of external hardware and software overhead.



210464-1

Figure 1. Block Diagram



210464-2

Figure 2. Pin Configuration

November 1986
Order Number: 210464-002

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
LC_3 LC_2 LC_1 LC_0	1 2 3 4	O	**LINE COUNT:** Output from the line counter which is used to address the character generator for the line positions on the screen.
DRQ	5	O	**DMA REQUEST:** Output signal to the 8257 DMA controller requesting a DMA cycle.
$\overline{DACK}$	6	I	**DMA ACKNOWLEDGE:** Input signal from the 8257H DMA controller acknowledging that the requested DMA cycle has been granted.
HRTC	7	O	**HORIZONTAL RETRACE:** Output signal which is active during the programmed horizontal retrace interval. During this period the VSP output is high and the LTEN output is low.
VRTC	8	O	**VERTICAL RETRACE:** Output signal which is active during the programmed vertical retrace interval. During this period the VSP output is high and the LTEN output is low.
$\overline{RD}$	9	I	**READ INPUT:** A control signal to read registers.
$\overline{WR}$	10	I	**WRITE INPUT:** A control signal to write commands into the control registers or write data into the row buffers during a DMA cycle.
LPEN	11	I	**LIGHT PEN:** Input signal from the CRT system signifying that a light pen signal has been detected.
DB_0 DB_1 DB_2 DB_3 DB_4 DB_5 DB_6 DB_7	12 13 14 15 16 17 18 19	I/O	**BI-DIRECTIONAL THREE-STATE DATA BUS LINES:** The outputs are enabled during a read of the C or P ports.
Ground	20		**GROUND.**
V_{CC}	40		**+5V POWER SUPPLY.**
LA_0 LA_1	39 38	O	**LINE ATTRIBUTE CODES:** These attribute codes have to be decoded externally by the dot/timing logic to generate the horizontal and vertical line combinations for the graphic displays specified by the character attribute codes.

Table 1. Pin Descriptions (Continued)

Symbol	Pin No.	Type	Name and Function
LTEN	37	O	**LIGHT ENABLE:** Output signal used to enable the video signal to the CRT. This output is active at the programmed underline cursor position, and at positions specified by attribute codes.
RVV	36	O	**REVERSE VIDEO:** Output signal used to indicate the CRT circuitry to reverse the video signal. This output is active at the cursor position if a reverse video block cursor is programmed or at the positions specified by the field attribute codes.
VSP	35	O	**VIDEO SUPPRESSION:** Output signal used to blank the video signal to the CRT. This output is active: • during the horizontal and vertical retrace intervals. • at the top and bottom lines of rows if underline is programmed to be number 8 or greater. • when an end of row or end of screen code is detected. • when a DMA underrun occurs. • at regular intervals ($1/16$ frame frequency for cursor, $1/32$ frame frequency for character and field attributes)—to create blinking displays as specified by cursor, character attribute, or field attribute programming.
GPA_1 GPA_0	34 33	O	**GENERAL PURPOSE ATTRIBUTE CODES:** Outputs which are enabled by the general purpose field attribute codes.
HLGT	32	O	**HIGHLIGHT:** Output signal used to intensify the display at particular positions on the screen as specified by the character attribute codes or field attribute codes.
IRQ	31	O	**INTERRUPT REQUEST.**
CCLK	30	I	**CHARACTER CLOCK (from Dot/Timing Logic).**
CC_6 CC_5 CC_4 CC_3 CC_2 CC_1 CC_0	29 28 27 26 25 24 23	O	**CHARACTER CODES:** Output from the row buffers used for character selection in the character generator.
CS	22	I	**CHIP SELECT:** The read and write are enabled by CS.
A_0	21	I	**PORT ADDRESS:** A high input on A_0 selects the "C" port or command registers and a low input selects the "P" port or parameter registers.

FUNCTIONAL DESCRIPTION

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8275H to the system Data Bus.

This functional block accepts inputs from the System Control Bus and generates control signals for overall device operation. It contains the Command, Parameter, and Status Registers that store the various control formats for the device functional definition.

A_0	Operation	Register
0	Read	PREG
0	Write	PREG
1	Read	SREG
1	Write	CREG

A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	1	0	0	Write 8275H Parameter
0	0	1	0	Read 8275H Parameter
1	1	0	0	Write 8275H Command
1	0	1	0	Read 8275H Status
X	1	1	0	Three-State
X	X	X	1	Three-State

$\overline{RD}$ (READ)

A "low" on this input informs the 8275H that the CPU is reading data or status information from the 8275H.

$\overline{WR}$ (WRITE)

A "low" on this input informs the 8275H that the CPU is writing data or control words to the 8275H.

$\overline{CS}$ (CHIP SELECT)

A "low" on this input selects the 8275H. No reading or writing will occur unless the device is selected. When $\overline{CS}$ is high, the Data Bus is in the float state and $\overline{RD}$ and $\overline{WR}$ will have no effect on the chip.

DRQ (DMA REQUEST)

A "high" on this output informs the DMA Controller that the 8275H desires a DMA transfer.

$\overline{DACK}$ (DMA ACKNOWLEDGE)

A "low" on this input informs the 8275H that a DMA cycle is in progress.

IRQ (INTERRUPT REQUEST)

A "high" on this output informs the CPU that the 8275H desires interrupt service.

Character Counter

The Character Counter is a programmable counter that is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. It is driven by the CCLK (Character Clock) input, which should be a derivative of the external dot clock.

Line Counter

The Line Counter is a programmable counter that is used to determine the number of horizontal lines (Sweeps) per character row. Its outputs are used to address the external character generator ROM.

Row Counter

The Row Counter is a programmable counter that is used to determine the number of character rows to be displayed per frame and length of the vertical retrace interval.

Light Pen Registers

The Light Pen Registers are two registers that store the contents of the character counter and the row counter whenever there is a rising edge on the LPEN (Light Pen) input.

NOTE:
Software correction is required.

Raster Timing and Video Controls

The Raster Timing circuitry controls the timing of the HRTC (Horizontal Retrace) and VRTC (Vertical Retrace) outputs. The Video Control circuitry controls the generation of LA_{0-1} (Line Attribute), HGLT (Highlight), RVV (Reverse Video), LTEN (Light Enable), VSP (Video Suppress), and GPA_{0-1} (General Purpose Attribute) outputs.

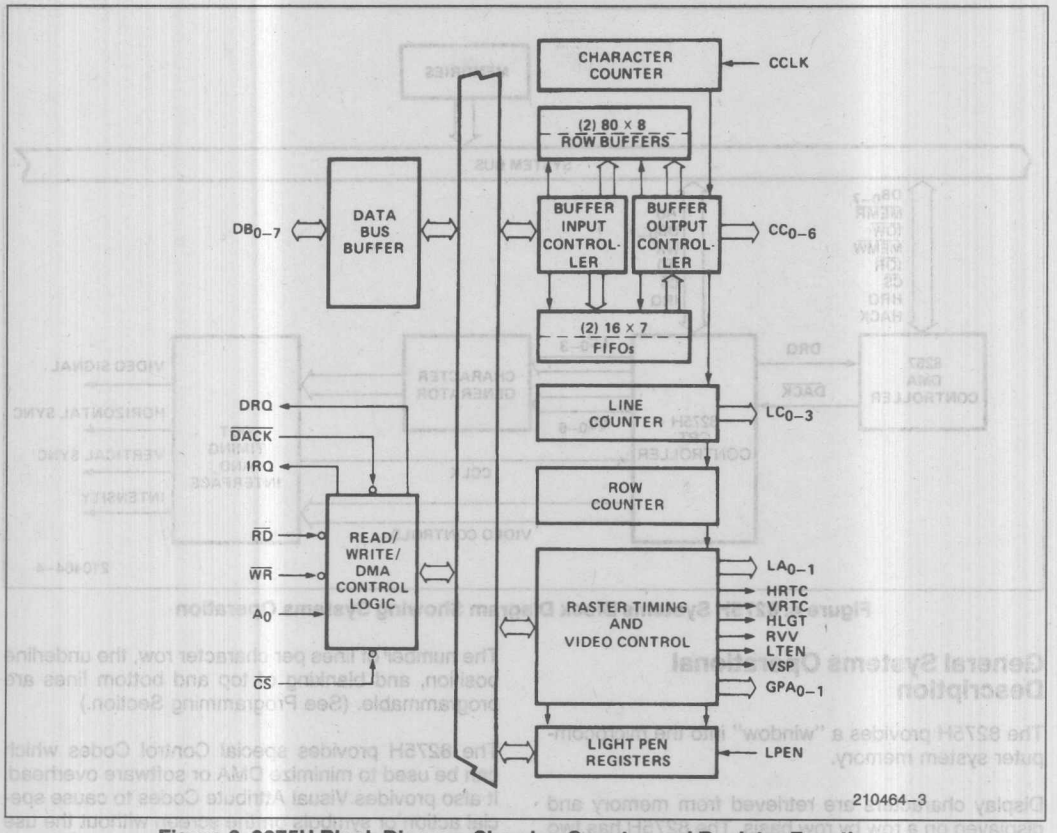


Figure 3. 8275H Block Diagram Showing Counter and Register Functions

Row Buffers

The Row Buffers are two 80 character buffers. They are filled from the microcomputer system memory with the character codes to be displayed. While one row buffer is displaying a row of characters, the other is being filled with the next row of characters.

FIFOs

There are two 16 character FIFOs in the 8275H. They are used to provide extra row buffer length in the Transparent Attribute Mode (see Detailed Operation section).

Buffer Input/Output Controllers

The Buffer Input/Output Controllers decode the characters being placed in the row buffers. If the character is a character attribute, field attribute or special code, these controllers control the appropriate action. (Examples: An "End of Screen—Stop DMA" special code will cause the Buffer Input Controller to stop further DMA requests. A "Highlight" field attribute will cause the Buffer Output Controller to activate the HGLT output.)

SYSTEM OPERATION

The 8275H is programmable to a large number of different display formats. It provides raster timing, display row buffering, visual attribute decoding, cursor timing, and light pen detection.

It is designed to interface with the 8257 DMA Controller and standard character generator ROMs for dot matrix decoding. Dot level timing must be provided by external circuitry.

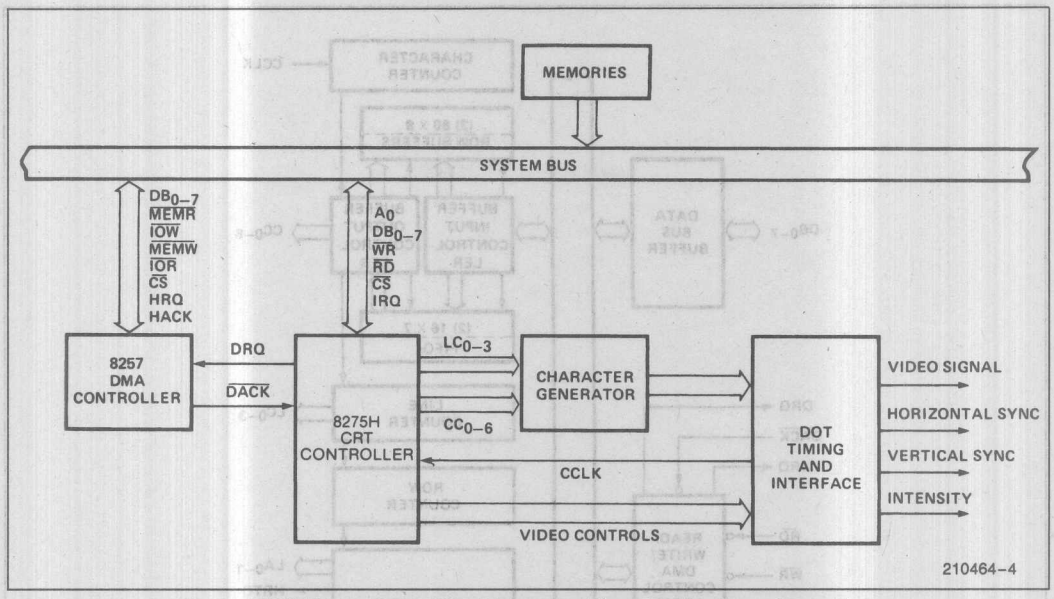


Figure 4. 8275H Systems Block Diagram Showing Systems Operation

General Systems Operational Description

The 8275H provides a "window" into the microcomputer system memory.

Display characters are retrieved from memory and displayed on a row by row basis. The 8275H has two row buffers. While one row buffer is being used for display, the other is being filled with the next row of characters to be displayed. The number of display characters per row and the number of character rows per frame are software programmable, providing easy interface to most CRT displays. (See Programming Section.)

The 8275H requests DMA to fill the row buffer that is not being used for display. DMA burst length and spacing is programmable. (See Programming Section.)

The 8275H displays character rows one line at a time.

The number of lines per character row, the underline position, and blanking of top and bottom lines are programmable. (See Programming Section.)

The 8275H provides special Control Codes which can be used to minimize DMA or software overhead. It also provides Visual Attribute Codes to cause special action or symbols on the screen without the use of the character generator (See Visual Attributes Section).

The 8275H also controls raster timing. This is done by generating Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) signals. The timing of these signals is programmable.

The 8275H can generate a cursor. Cursor location and format are programmable. (See Programming Section.)

The 8275H has a light pen input and registers. The light pen input is used to load the registers. Light pen registers can be read on command. (See Programming Section.)

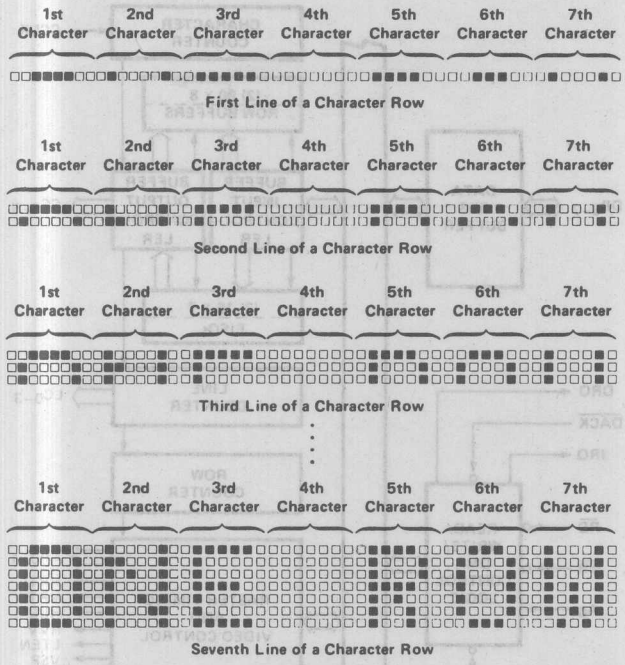


Figure 5. Display of a Character Row

Display Row Buffering

Before the start of a frame, the 8275H requests DMA and one row buffer is filled with characters.

When the first horizontal sweep is started, character codes are output to the character generator from the row buffer just filled. Simultaneously, DMA begins filling the other row buffer with the next row of characters.

After all the lines of the character row are scanned, the roles of the two row buffers are reversed and the same procedure is followed for the next row.

This is repeated until all of the character rows are displayed.

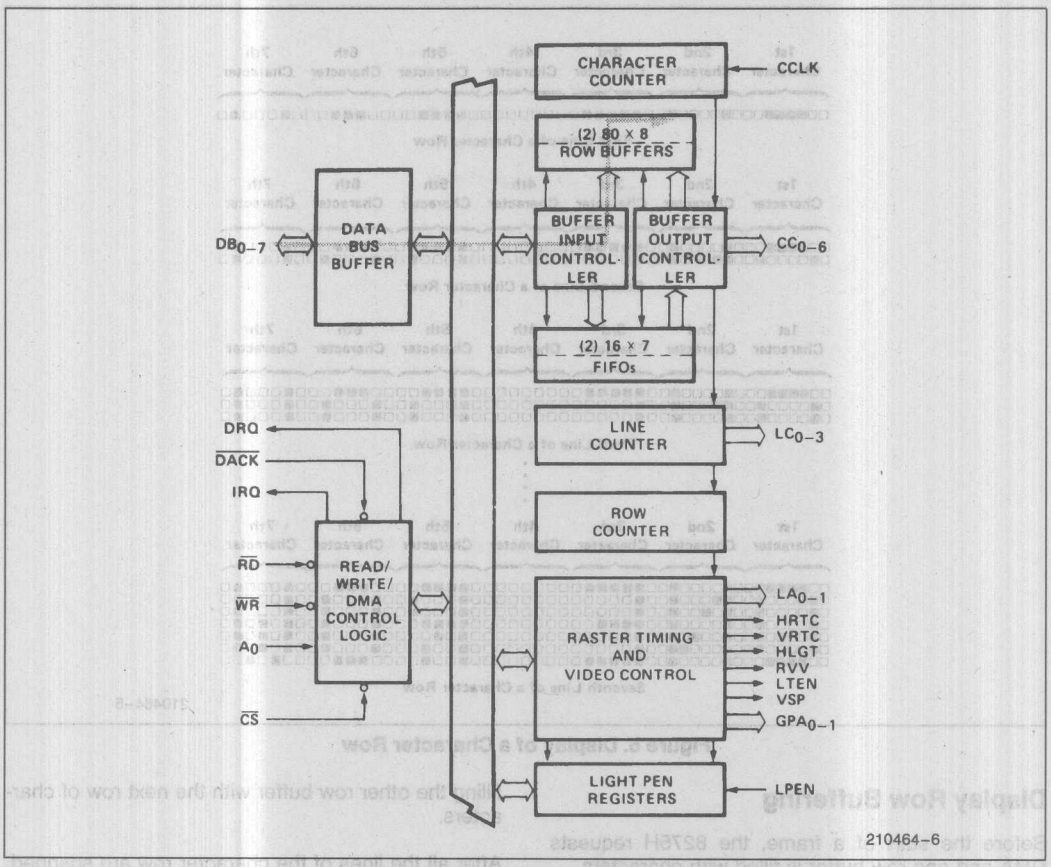


Figure 6. First Row Buffer Filled

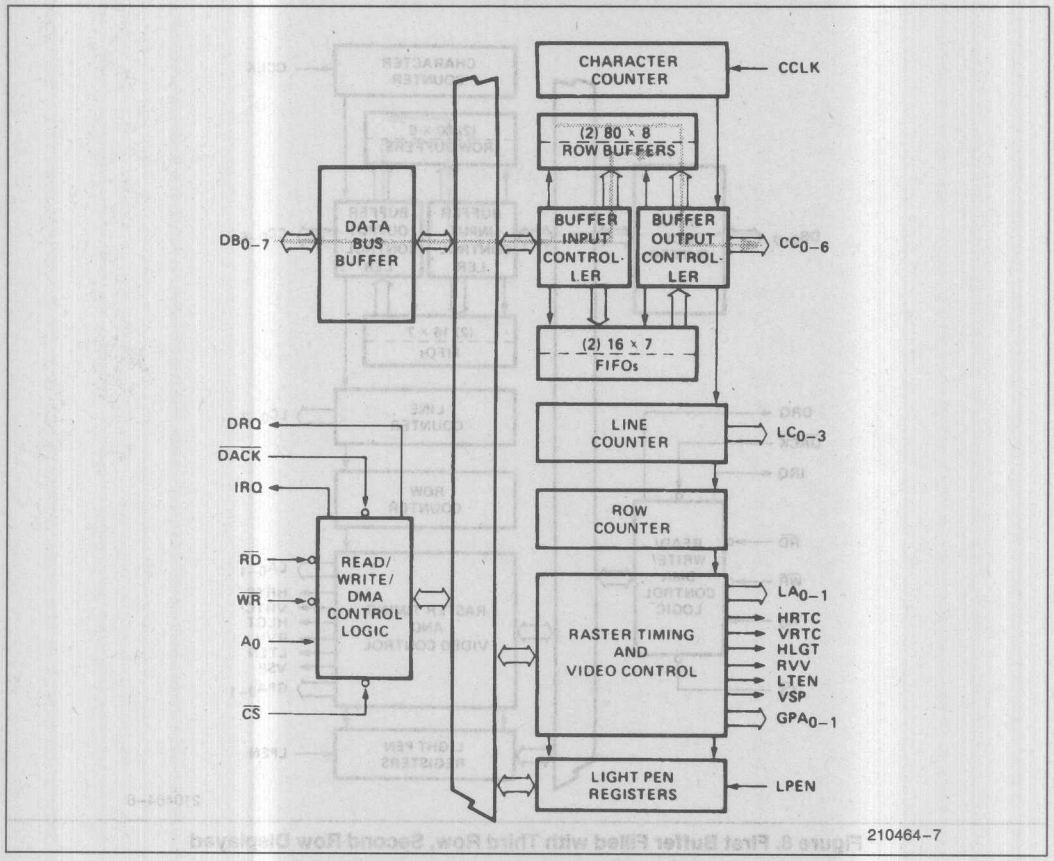


Figure 7. Second Buffer Filled, First Row Displayed

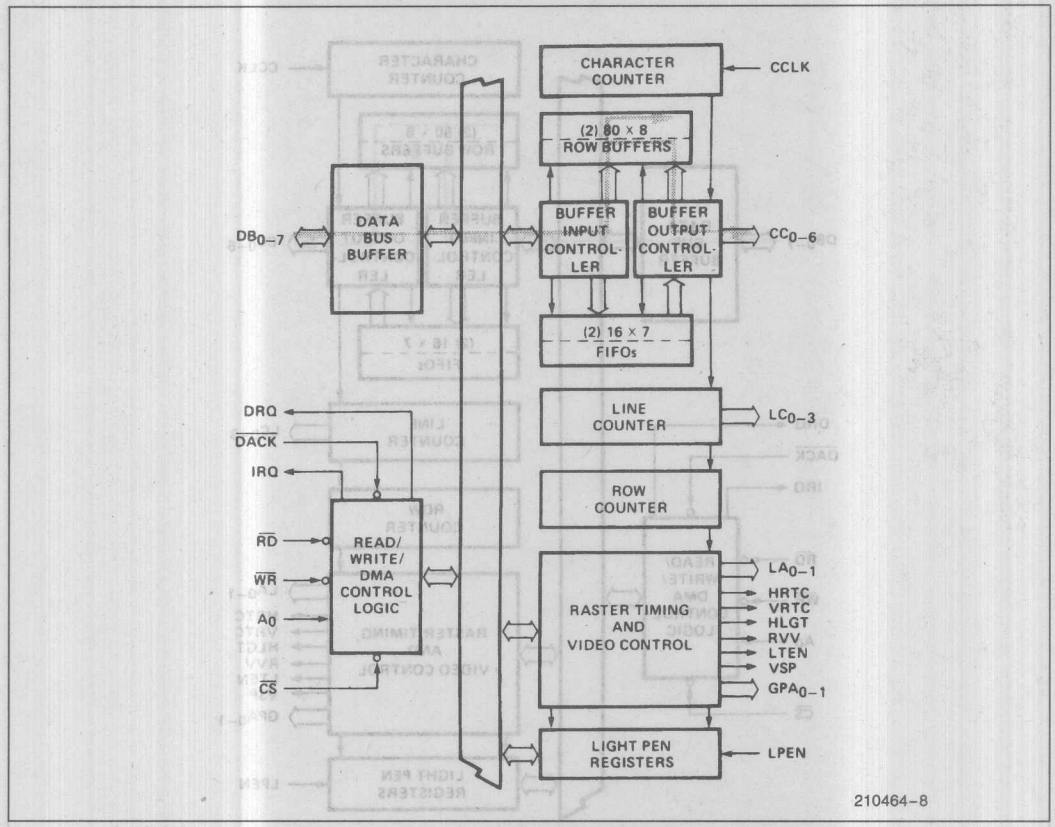


Figure 8. First Buffer Filled with Third Row, Second Row Displayed

Display Format

SCREEN FORMAT

The 8275H can be programmed to generate from 1 to 80 characters per row, and from 1 to 64 rows per frame.

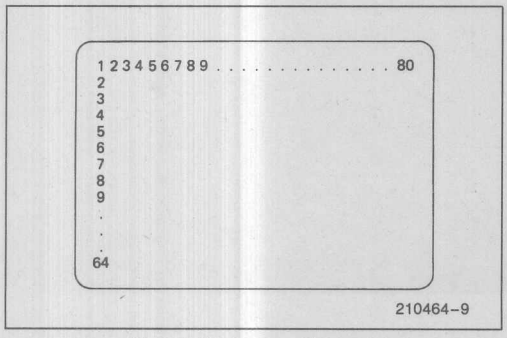


Figure 9. Screen Format

The 8275H can also be programmed to blank alternate rows. In this mode the first row is displayed, the second blanked, the third displayed, etc. DMA is not requested for the blanked rows.

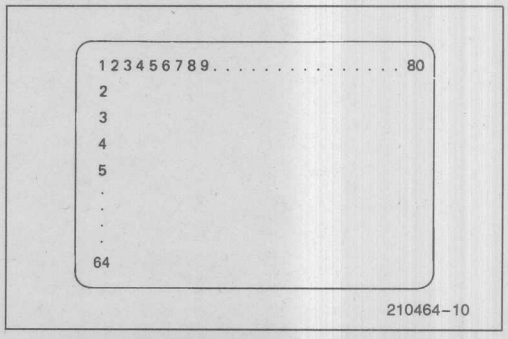


Figure 10. Blank Alternate Rows Mode

ROW FORMAT

The 8275H is designed to hold the line count stable while outputting the appropriate character codes during each horizontal sweep. The line count is incremented during horizontal retrace and the whole row of character codes are output again during the next sweep. This is continued until the whole character row is displayed.

The number of lines (horizontal sweeps) per character row is programmable from 1 to 16.

The output of the line counter can be programmed to be in one of two modes.

In mode 0, the output of the line *counter* is the same as the line *number*.

In mode 1, the line *counter* is offset by one from the line *number*.

NOTE:
In mode 1, while the *first* line (line number 0) is being displayed, the *last* count is output by the line counter (see examples).

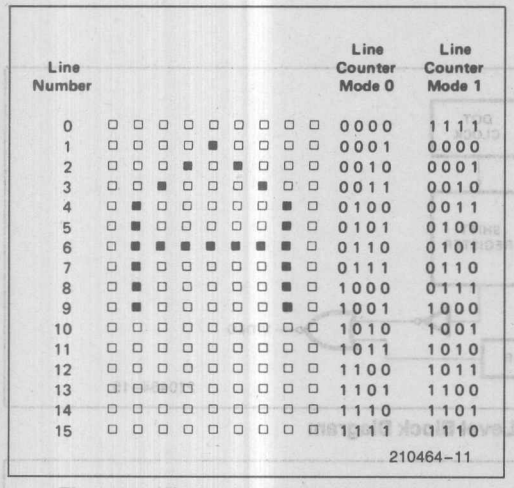


Figure 11. Example of a 16-Line Format

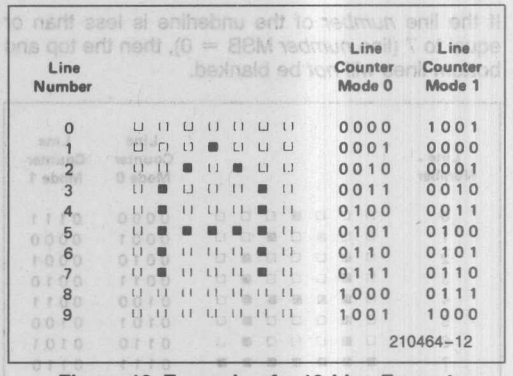


Figure 12. Example of a 10-Line Format

Mode 0 is useful for character generators that leave address zero blank and start at address 1. Mode 1 is useful for character generators which start at address zero.

Underline placement is also programmable (from line *number* 0 to 15). This is independent of the line *counter* mode.

If the line *number* of the underline is greater than 7 (line *number* MSB = 1), then the top and bottom lines will be blanked.

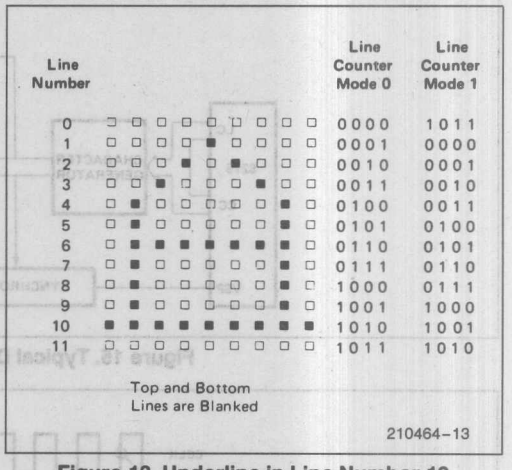


Figure 13. Underline in Line Number 10

If the line *number* of the underline is less than or equal to 7 (line *number* MSB = 0), then the top and bottom lines will *not* be blanked.

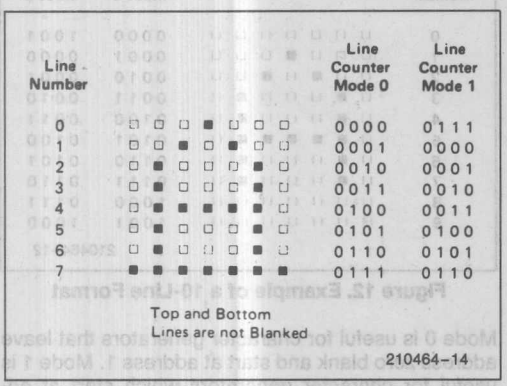


Figure 14. Underline in Line Number 7

If the line *number* of the underline is greater than the maximum number of lines, the underline will not appear.

Blanking is accomplished by the VSP (Video Suppression) signal. Underline is accomplished by the LTEN (Light Enable) signal.

DOT FORMAT

Dot width and character width are dependent upon the external timing and control circuitry.

Dot level timing circuitry should be designed to accept the parallel output of the character generator and shift it out serially at the rate required by the CRT display.

Dot width is a function of dot clock frequency

Character width is a function of the character generator width.

Horizontal character spacing is a function of the shift register length.

NOTE:
Video control and timing signals must be synchronized with the video signal due to the character generator access delay.

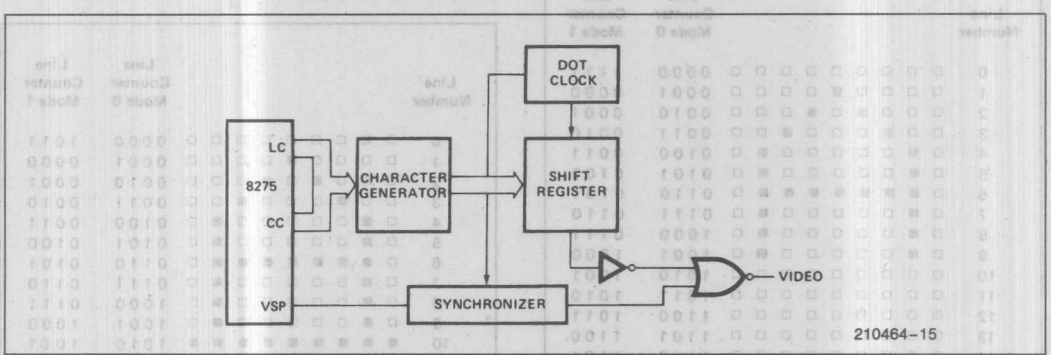


Figure 15. Typical Dot Level Block Diagram

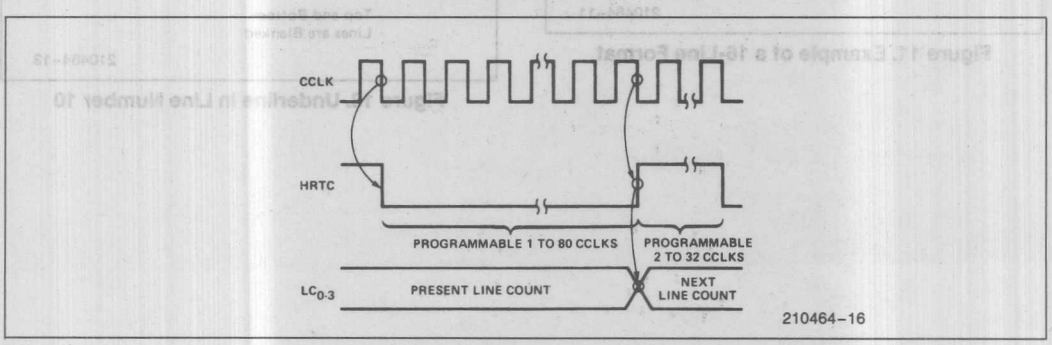


Figure 16. Line Timing

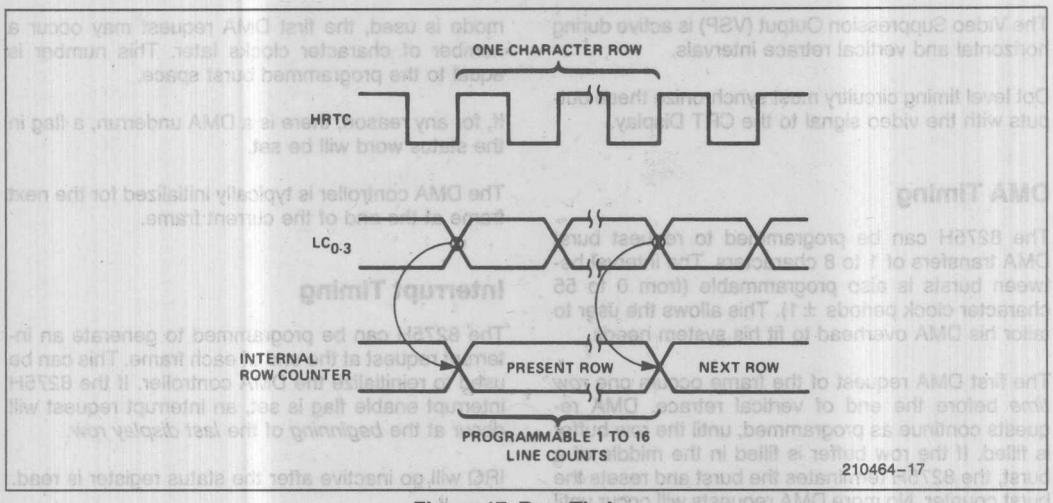


Figure 17. Row Timing

Raster Timing

The character counter is driven by the character clock input (CCLK). It counts out the characters being displayed (programmable from 1 to 80). It then causes the line counter to increment, and it starts counting out the horizontal retrace interval (programmable from 2 to 32). This is constantly repeated.

The line counter is driven by the character counter. It is used to generate the line address outputs (LC_{0-3}) for the character generator. After it counts all of the lines in a character row (programmable from 1 to 16), it increments the row counter, and starts over again. (See Character Format Section for detailed description of Line Counter functions.)

The row counter is an internal counter driven by the line counter. It controls the functions of the row buffers and counts the number of character rows displayed.

After the row counter counts all of the rows in a frame (programmable from 1 to 64), it starts counting out the vertical retrace interval (programmable from 1 to 4).

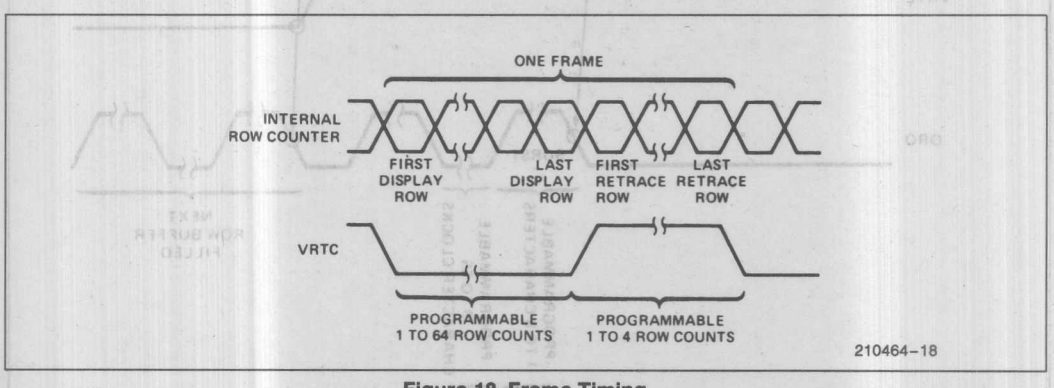


Figure 18. Frame Timing

The Video Suppression Output (VSP) is active during horizontal and vertical retrace intervals.

Dot level timing circuitry must synchronize these outputs with the video signal to the CRT Display.

DMA Timing

The 8275H can be programmed to request burst DMA transfers of 1 to 8 characters. The interval between bursts is also programmable (from 0 to 55 character clock periods ±1). This allows the user to tailor his DMA overhead to fit his system needs.

The first DMA request of the frame occurs one *row time* before the end of vertical retrace. DMA requests continue as programmed, until the row buffer is filled. If the row buffer is filled in the middle of a burst, the 8275H terminates the burst and resets the burst counter. No more DMA requests will occur until the *beginning* of the *next row*. At that time, DMA requests are activated as programmed until the other buffer is filled.

The first DMA request for a row will start at the first character clock of the preceding row. If the burst mode is used, the first DMA request may occur a number of character clocks later. This number is equal to the programmed burst space.

If, for any reason, there is a DMA underrun, a flag in the status word will be set.

The DMA controller is typically initialized for the next frame at the end of the current frame.

Interrupt Timing

The 8275H can be programmed to generate an interrupt request at the end of each frame. This can be used to reinitialize the DMA controller. If the 8275H interrupt enable flag is set, an interrupt request will occur at the *beginning* of the *last display row*.

IRQ will go inactive after the status register is read.

A reset command will also cause IRQ to go inactive, but this is not recommended during normal service.

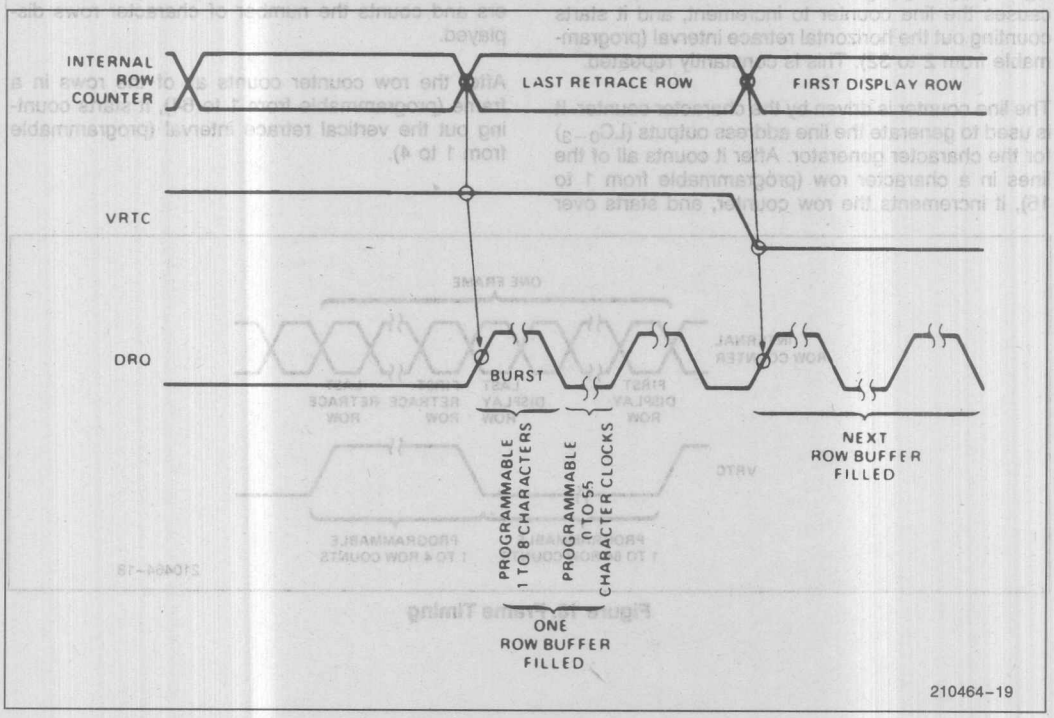


Figure 19. DMA Timing

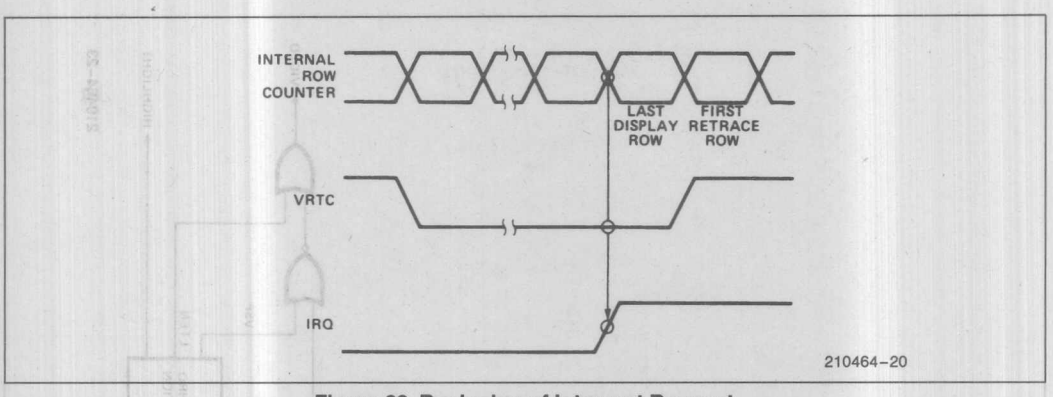


Figure 20. Beginning of Interrupt Request

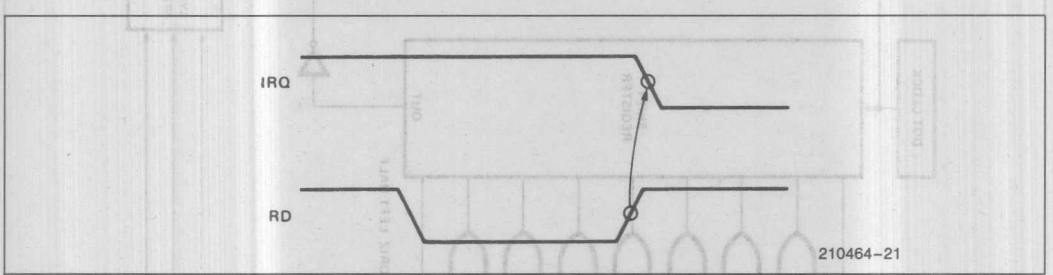


Figure 21. End of Interrupt Request

Another method of reinitializing the DMA controller is to have the DMA controller itself interrupt on terminal count. With this method, the 8275H interrupt enable flag should not be set.

NOTE:
Upon power-up, the 8275H Interrupt Enable Flag may be set. As a result, the user's cold start routine should write a reset command to the 8275H before system interrupts are enabled.

VISUAL ATTRIBUTES AND SPECIAL CODES

The characters processed by the 8275H are 8-bit quantities. The character code outputs provide the character generator with 7 bits of address. The Most Significant Bit is the extra bit and it is used to determine if it is a normal display character (MSB = 0), or if it is a Visual Attribute or Special Code (MSB = 1).

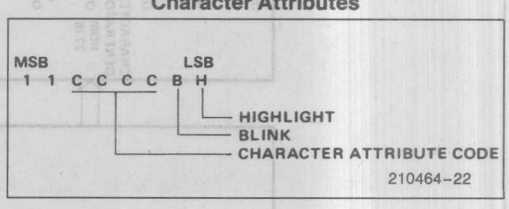
There are two types of Visual Attribute Codes. They are Character Attributes and Field Attributes.

CHARACTER ATTRIBUTE CODES

Character attribute codes are codes that can be used to generate graphics symbols without the use of a character generator. This is accomplished by selectively activating the Line Attribute outputs (LA_{0-1}), the Video Suppression output (VSP), and the Light Enable output. The dot level timing circuitry can use these signals to generate the proper symbols.

Character attributes can be programmed to blink or be highlighted individually. Blinking is accomplished with the Video Suppression output (VSP). Blink frequency is equal to the screen refresh frequency divided by 32. Highlighting is accomplished by activating the Highlight output (HGLT).

Character Attributes



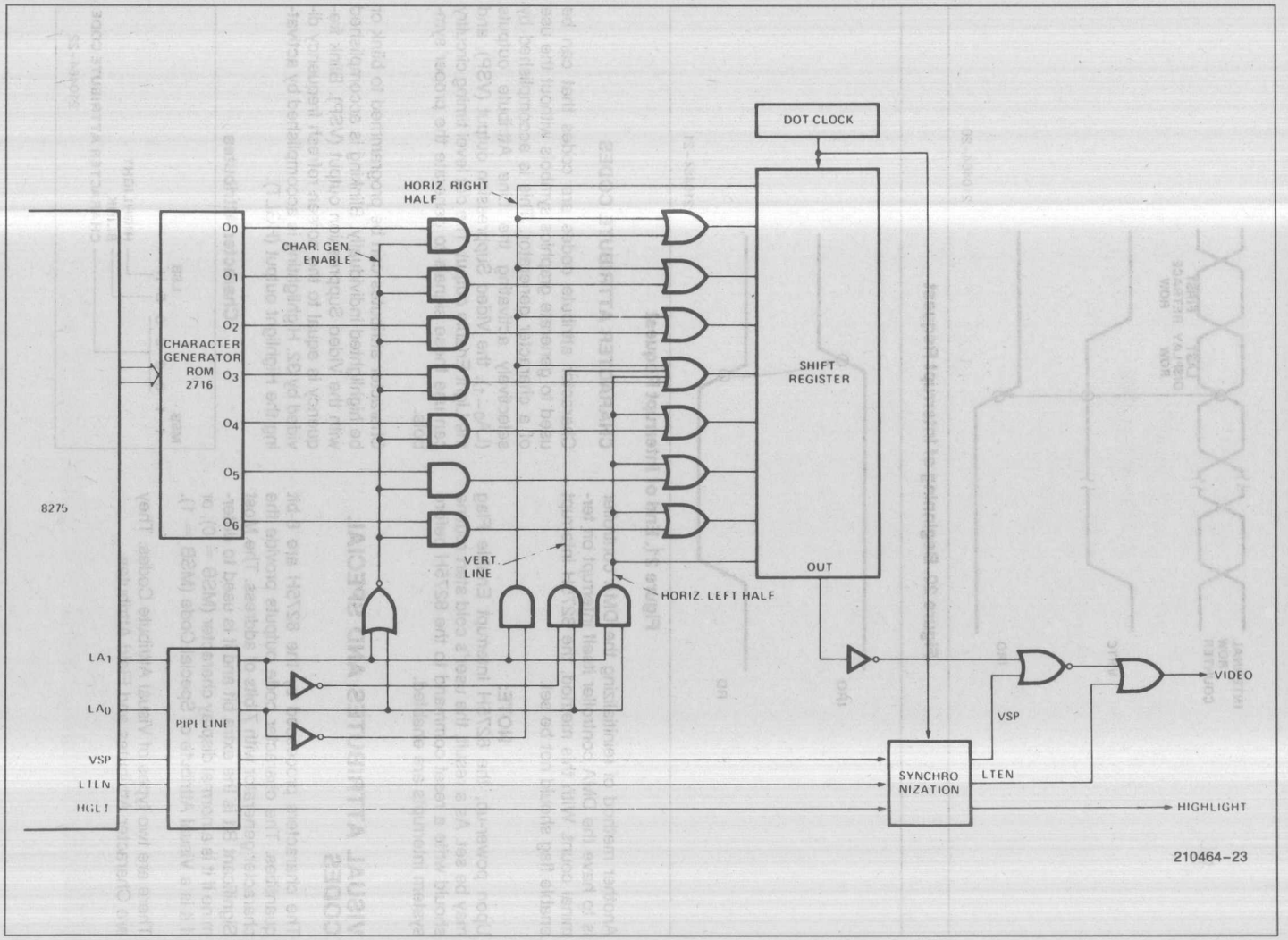


Figure 22. Typical Character Attribute Logic

Table 2. Character Attributes

Character attributes were designed to produce the following graphics:

CHARACTER ATTRIBUTE CODE "CCCC"		OUTPUTS LA_1	LA_0	VSP	LTEN	SYMBOL	DESCRIPTION
0000	Above Underline	0	0	1	0		Top Left Corner
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0001	Above Underline	0	0	1	0		Top Right Corner
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0010	Above Underline	0	0	1	0		Bottom Left Corner
	Underline	1	0	0	0		
	Below Underline	0	0	1	0		
0011	Above Underline	0	1	0	0		Bottom Right Corner
	Underline	1	1	0	0		
	Below Underline	0	0	1	0		
0100	Above Underline	0	0	1	0		Top Intersect
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
0101	Above Underline	0	1	0	0		Right Intersect
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0110	Above Underline	0	1	0	0		Left Intersect
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0111	Above Underline	0	1	0	0		Bottom Intersect
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1000	Above Underline	0	0	1	0		Horizontal Line
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1001	Above Underline	0	1	0	0		Vertical Line
	Underline	0	1	0	0		
	Below Underline	0	1	0	0		
1010	Above Underline	0	1	0	0		Crossed Lines
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
1011	Above Underline	0	0	0	0		Not Recommended *
	Underline	0	0	0	0		
	Below Underline	0	0	0	0		
1100	Above Underline	0	0	1	0		Special Codes
	Underline	0	0	1	0		
	Below Underline	0	0	1	0		
1101	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1110	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1111	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						

210464-24

*Character Attribute Code 1011 is not recommended for normal operation. Since none of the attribute outputs are active, the character Generator will not be disabled, and an indeterminate character will be generated.

Character Attribute Codes 1101, 1110, and 1111 are illegal.

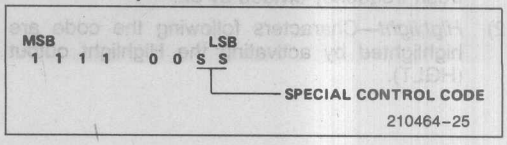
Blinking is active when B = 1.

Highlight is active when H = 1.

SPECIAL CODES

Four special codes are available to help reduce memory, software, or DMA overhead.

Special Control Character



210464-25

S	S	Function
0	0	End of Row
0	1	End of Row-Stop DMA
1	0	End of Screen
1	1	End of Screen-Stop DMA

The End of Row Code (00) activates VSP and holds it to the end of the line.

The End of Row-Stop DMA Code (01) causes the DMA Control Logic to stop DMA for the rest of the row when it is written into the Row Buffer. It affects the display in the same way as the End of Row Code (00).

The End of Screen Code (10) activates VSP and holds it to the end of the frame.

The End of Screen-Stop DMA Code (11) causes the DMA Control Logic to stop DMA for the rest of the frame when it is written into the Row Buffer. It affects the display in the same way as the End of Screen Code (10).

If the Stop DMA feature is not used, all characters after an End of Row character are ignored, except for the End of Screen character, which operates normally. All characters after an End of Screen character are ignored.

NOTE:
If a Stop DMA character is not the last character in a burst or row, DMA is not stopped until after the next character is read. In this situation, a dummy character must be placed in memory after the Stop DMA character.

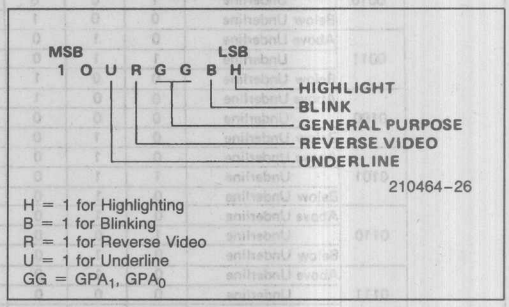
FIELD ATTRIBUTES

The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the code up to, and including, the character which precedes the *next* field attribute code, or up to the end of the frame. The field attributes are reset during the vertical retrace interval.

There are six field attributes:

1) *Blink*—Characters following the code blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2) *Highlight*—Characters following the code are highlighted by activating the Highlight output (HGLT).
3) *Reverse Video*—Characters following the code appear with reverse video by activating the Reverse Video output (RVV).
4) *Underline*—Characters following the code are underlined by activating the Light Enable output (LTEN).
5,6) *General Purpose*—There are two additional 8275H outputs which act as general purpose, independently programmable field attributes. GPA_{0-1} are active high outputs.

Field Attribute Code



NOTE:
More than one attribute can be enabled at the same time. If the blinking and reverse video attributes are enabled simultaneously, only the reversed characters will blink.

The 8275H can be programmed to provide visible or invisible field attribute characters.

If the 8275H is programmed in the visible field attribute mode, all field attributes will occupy a position on the screen. They will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character.



Figure 23. Example of the Visible Field Attribute Mode (Underline Attribute)

If the 8275H is programmed in the invisible field attribute mode, the 8275H FIFO is activated.

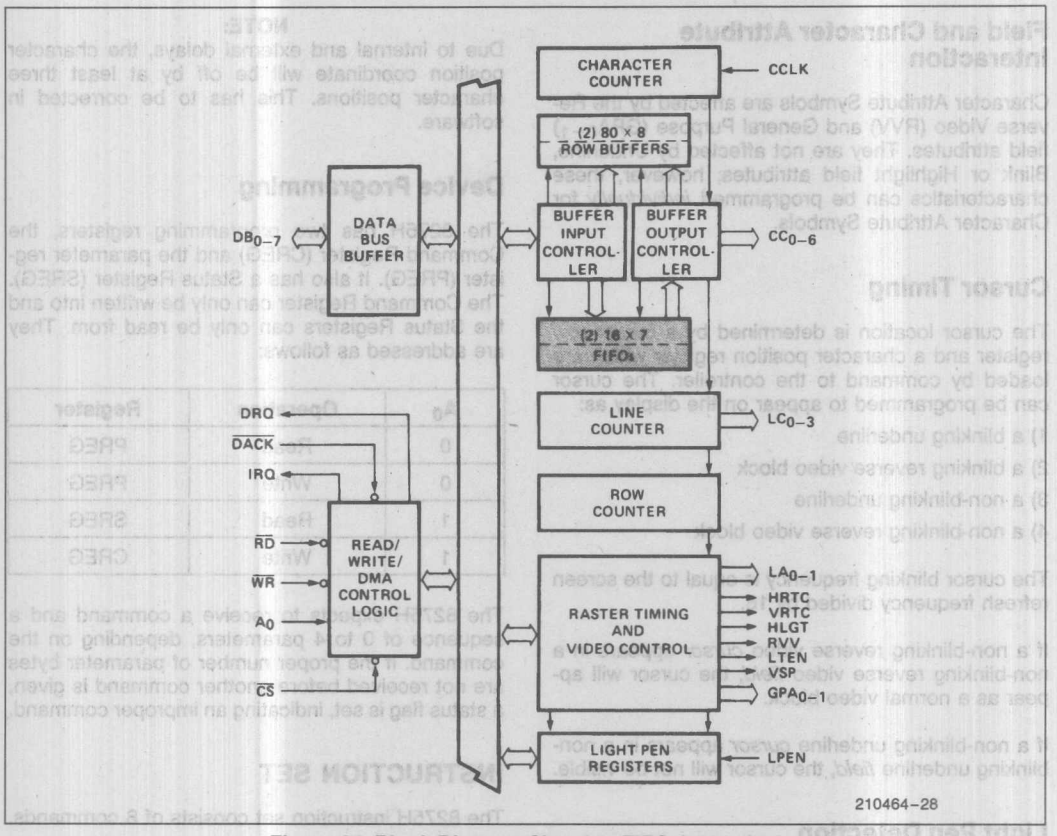


Figure 24. Block Diagram Showing FIFO Activation

Each row buffer has a corresponding FIFO. These FIFOs are 16 characters by 7 bits in size.

When a field attribute is placed in the row buffer during DMA, the buffer input controller recognizes it and places the *next* character in the proper FIFO.

When a field attribute is placed in the Buffer Output Controller during display, it causes the controller to immediately put a character from the FIFO on the Character Code outputs (CC_{0-6}). The chosen Visual Attributes are also activated.

Since the FIFO is 16 characters long, no more than 16 field attribute characters may be used per line in this mode. If more are used, a bit in the status word is set and the first characters in the FIFO are written over and lost.

NOTE:
Since the FIFO is 7 bits wide, the MSB of any characters put in it are stripped off. Therefore, a Visual Attribute or Special Code must *not* immediately follow a field attribute code. If this situation does occur, the Visual Attribute or Special Code will be treated as a normal display character.

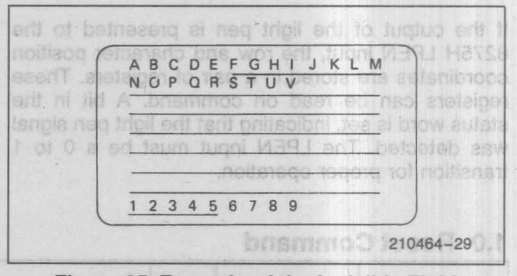


Figure 25. Example of the Invisible Field Attribute Mode (Underline Attribute)

Field and Character Attribute Interaction

Character Attribute Symbols are affected by the Reverse Video (RVV) and General Purpose (GPA_{0-1}) field attributes. They are not affected by Underline, Blink or Highlight field attributes; however, these characteristics can be programmed *individually* for Character Attribute Symbols.

Cursor Timing

The cursor location is determined by a cursor row register and a character position register which are loaded by command to the controller. The cursor can be programmed to appear on the display as:

1) a blinking underline
2) a blinking reverse video block
3) a non-blinking underline
4) a non-blinking reverse video block

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

If a non-blinking reverse video *cursor* appears in a non-blinking reverse video *field*, the cursor will appear as a normal video block.

If a non-blinking underline *cursor* appears in a non-blinking underline *field*, the cursor will not be visible.

Light Pen Detection

A light pen consists of a micro switch and a tiny light sensor. When the light pen is pressed against the CRT screen, the micro switch enables the light sensor. When the raster sweep reaches the light sensor, it triggers the light pen output.

If the output of the light pen is presented to the 8275H LPEN input, the row and character position coordinates are stored in a pair of registers. These registers can be read on command. A bit in the status word is set, indicating that the light pen signal was detected. The LPEN input must be a 0 to 1 transition for proper operation.

NOTE:
Due to internal and external delays, the character position coordinate will be off by at least three character positions. This has to be corrected in software.

Device Programming

The 8275H has two programming registers, the Command Register (CREG) and the parameter register (PREG). It also has a Status Register (SREG). The Command Register can only be written into and the Status Registers can only be read from. They are addressed as follows:

A_0	Operation	Register
0	Read	PREG
0	Write	PREG
1	Read	SREG
1	Write	CREG

The 8275H expects to receive a command and a sequence of 0 to 4 parameters, depending on the command. If the proper number of parameter bytes are not received before another command is given, a status flag is set, indicating an improper command.

INSTRUCTION SET

The 8275H instruction set consists of 8 commands.

Command	No. of Parameter Bytes
Reset	4
Start Display	0
Stop Display	0
Read Light Pen	2
Load Cursor	2
Enable Interrupt	0
Disable Interrupt	0
Preset Counters	0

In addition, the status of the 8275H (SREG) can be read by the CPU at any time.

1.0 Reset Command

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Write	1	Reset Command	0	0	0	0	0	0	0	0
Parameters	Write	0	Screen Comp Byte 1	S	H	H	H	H	H	H	H
	Write	0	Screen Comp Byte 2	V	V	R	R	R	R	R	R
	Write	0	Screen Comp Byte 3	U	U	U	U	L	L	L	L
	Write	0	Screen Comp Byte 4	M	F	C	C	Z	Z	Z	Z

Action—After the reset command is written, DMA requests stop, 8275H interrupts are disabled, and the VSP output is used to blank the screen. HRTC and VRTC continue to run. HRTC and VRTC timing are random on power-up.

As parameters are written, the screen composition is defined.

Parameter—S Spaced Rows

S	Functions
0	Normal Rows
1	Spaced Rows

Parameter—HHHHHHH
Horizontal Characters/Row

H	H	H	H	H	H	H	No. of Characters Per Row
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	2
0	0	0	0	0	1	0	3
			.				.
			.				.
			.				.
1	0	0	1	1	1	1	80
1	0	1	0	0	0	0	Undefined
			.				.
			.				.
			.				.
1	1	1	1	1	1	1	Undefined

Parameter—VV Vertical Retrace Row Count

V	V	No. of Row Counts Per VRTC
0	0	1
0	1	2
1	0	3
1	1	4

Parameter—RRRRRR Vertical Rows/Frame

R	R	R	R	R	R	No. of Rows/Frame
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
			.			.
			.			.
			.			.
1	1	1	1	1	1	64

Parameter—UUUU Underline Placement

U	U	U	U	Line Number of Underline
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
		.		.
		.		.
		.		.
1	1	1	1	16

Parameter—LLLL
Number of Lines per Character Row

L	L	L	L	No. of Lines/Row
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
		.		.
		.		.
		.		.
1	1	1	1	16

Parameter—M Line Counter Mode

M	Line Counter Mode
0	Mode 0 (Non-Offset)
1	Mode 1 (Offset by 1 Count)

Parameter—F Field Attribute Mode

F	Field Attribute Mode
0	Transparent
1	Non-Transparent

Parameter—CC Cursor Format

C	C	Cursor Format
0	0	Blinking reverse video block
0	1	Blinking underline
1	0	Nonblinking reverse video block
1	1	Nonblinking underline

Parameter—ZZZZ Horizontal Retrace Count

Z	Z	Z	Z	No. of Character Counts Per HRTC
0	0	0	0	2
0	0	0	1	4
0	0	1	0	6
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
1	1	1	1	32

NOTE:
uuuu MSB determines blanking of top and bottom lines (1 = blanked, 0 = not blanked).

SSS Burst Space Code

S	S	S	No. of Character Clocks Between DMA Requests
0	0	0	0
0	0	1	7
0	1	0	15
0	1	1	23
1	0	0	31
1	0	1	39
1	1	0	47
1	1	1	55

BB Burst Count Code

B	B	No. of DMA Cycles Per Burst
0	0	1
0	1	2
1	0	4
1	1	8

Action—8275H interrupts are enabled, DMA requests begin, video is enabled, Interrupt Enable and Video Enable status flags are set.

2.0 Start Display Command

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Write	1	Start Display	0	0	1	S	S	S	B	B
No Parameters											

3.0 Stop Display Command

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Write	1	Stop Display	0	1	0	0	0	0	0	0
No Parameters											

Action—Disables video, interrupts remain enabled, HRTC and VRTC continue to run, Video Enable status flag is reset, and the "Start Display" command must be given to re-enable the display.

4.0 Read Light Pen Command

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Write	1	Read Light Pen	0	1	1	0	0	0	0	0
Parameters	Read	0	Char. Number	(Char. Position in Row)							
	Read	0	Row Number	(Row Number)							

Action—The 8275H is conditioned to supply the contents of the light pen position registers in the next two read cycles of the parameter register. Status flags are not affected.

NOTE:
Software correction of light pen position is required.

5.0 Load Cursor Position

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Write	1	Load Cursor	1	0	0	0	0	0	0	0
Parameters	Write Write	0 0	Char. Number Row Number	(Char. Position in Row) (Row Number)							

Action—The 8275H is conditioned to place the next two parameter bytes into the cursor position registers. Status flags not affected.

6.0 Enable Interrupt Command

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Write	1	Enable Interrupt	1	0	1	0	0	0	0	0
No Parameters											

Action—The interrupt enable status flag is set and interrupts are enabled.

7.0 Disable Interrupt Command

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Write	1	Disable Interrupt	1	1	0	0	0	0	0	0
No Parameters											

Action—Interrupts are disabled and the interrupt enable status flag is reset.

8.0 Preset Counters Command

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Write	1	Preset Counters	1	1	1	0	0	0	0	0
No Parameters											

Action—The internal timing counters are preset, corresponding to a screen display position at the top left corner. Two character clocks are required for this operation. The counters will remain in this state until any other command is given.

This command is useful for system debug and synchronization of clustered CRT displays on a single CPU. After this command, two additional clock cycles are required before the first character of the first row is put out.

STATUS FLAGS

	Operation	A_0	Description	Data Bus MSB							LSB
Command	Read	1	Status Word	0	IE	IR	LP	IC	VE	DU	F0

IE — (Interrupt Enable) Set or reset by command. It enables vertical retrace interrupt. It is automatically set by a "Start Display" command and reset with the "Reset" command.

IR — (Interrupt Request) This flag is set at the beginning of display of the last row of the frame if the interrupt enable flag is set. It is reset after a status read operation.

LP — This flag is set when the light pen input (LPEN) is activated and the light pen registers have been loaded. This flag is automatically reset after a status read.

IC — (Improper Command) This flag is set when a command parameter string is too long or too short. The flag is automatically reset after a status read.

VE — (Video Enable) This flag indicates that video operation of the CRT is enabled. This flag is set on a "Start Display" command, and reset on a "Stop Display" or "Reset" command.

DU — (DMA Underrun) This flag is set whenever a data underrun occurs during DMA transfers. Upon detection of DU, the DMA operation is stopped and the screen is blanked until after the vertical retrace interval. This flag is reset after a status read.

FO — (FIFO Overrun) This flag is set whenever the FIFO is overrun. It is reset on a status read.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C

Storage Temperature ..... −65°C to +150°C

Voltage On Any Pin
With Respect to Ground ..... −0.5V to +7V

Power Dissipation ..... 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	

CAPACITANCE $T_A = 25°C$, $V_{CC} = GND = 0V$

Symbol	Parameter	Min	Max	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	$f_c = 1$ MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5.0V \pm 5\%$, $GND = 0V$

Bus Parameters

READ CYCLE

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{AR}	Address Stable before READ	0		ns	
t_{RA}	Address Hold Time for READ	0		ns	
t_{RR}	READ Pulse Width	250		ns	
t_{RD}	Data Delay from READ		200	ns	$C_L = 150$ pF
t_{DF}	READ to Data Floating		100	ns	$C_L = 150$ pF

WRITE CYCLE

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{AW}	Address Stable before WRITE	0		ns	
t_{WA}	Address Hold Time for WRITE	0		ns	
t_{WW}	WRITE Pulse Width	250		ns	
t_{DW}	Data Setup Time for WRITE	150		ns	
t_{WD}	Data Hold Time for WRITE	0		ns	

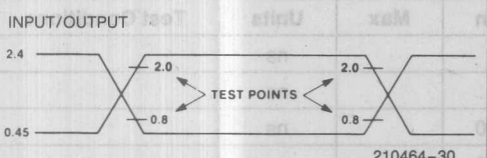
CLOCK TIMING

Symbol	Parameter	8275		8275-2		Units	Test Conditions
		Min	Max	Min	Max		
t_{CLK}	Clock Period	480		320		ns	
t_{KH}	Clock High	240		120		ns	
t_{KL}	Clock Low	160		120		ns	
t_{KR}	Clock Rise	5	30	5	30	ns	
t_{KF}	Clock Fall	5	30	5	30	ns	

OTHER TIMING

Symbol	Parameter	8275		8275-2		Units	Test Conditions
		Min	Max	Min	Max		
t_{CC}	Character Code Output Delay		150		150	ns	C_L = 50 pF
t_{HR}	Horizontal Retrace Output Delay		200		150	ns	C_L = 50 pF
t_{LC}	Line Count Output Delay		400		250	ns	C_L = 50 pF
t_{AT}	Control/Attribute Output Delay		275		250	ns	C_L = 50 pF
t_{VR}	Vertical Retrace Output Delay		275		250	ns	C_L = 50 pF
t_{RI}	IRQ ↓ from RD ↑		250		250	ns	C_L = 50 pF
t_{WQ}	DRQ ↑ from WR ↑		250		250	ns	C_L = 50 pF
t_{RQ}	DRQ ↓ from WR ↓		200		200	ns	C_L = 50 pF
t_{LR}	DACK ↓ to WR ↓	0		0		ns	
t_{RL}	WR ↑ to DACK ↑	0		0		ns	
t_{PR}	LPEN Rise		50		50	ns	
t_{PH}	LPEN Hold	100		100		ns	
t_{DI}	DACK Inactive Period	120				ns	

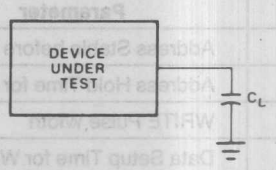
A.C. TESTING INPUT, OUTPUT WAVEFORM



210464-30

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0." Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0."

A.C. TESTING LOAD CIRCUIT



210464-31

C_L Includes Jig Capacitance

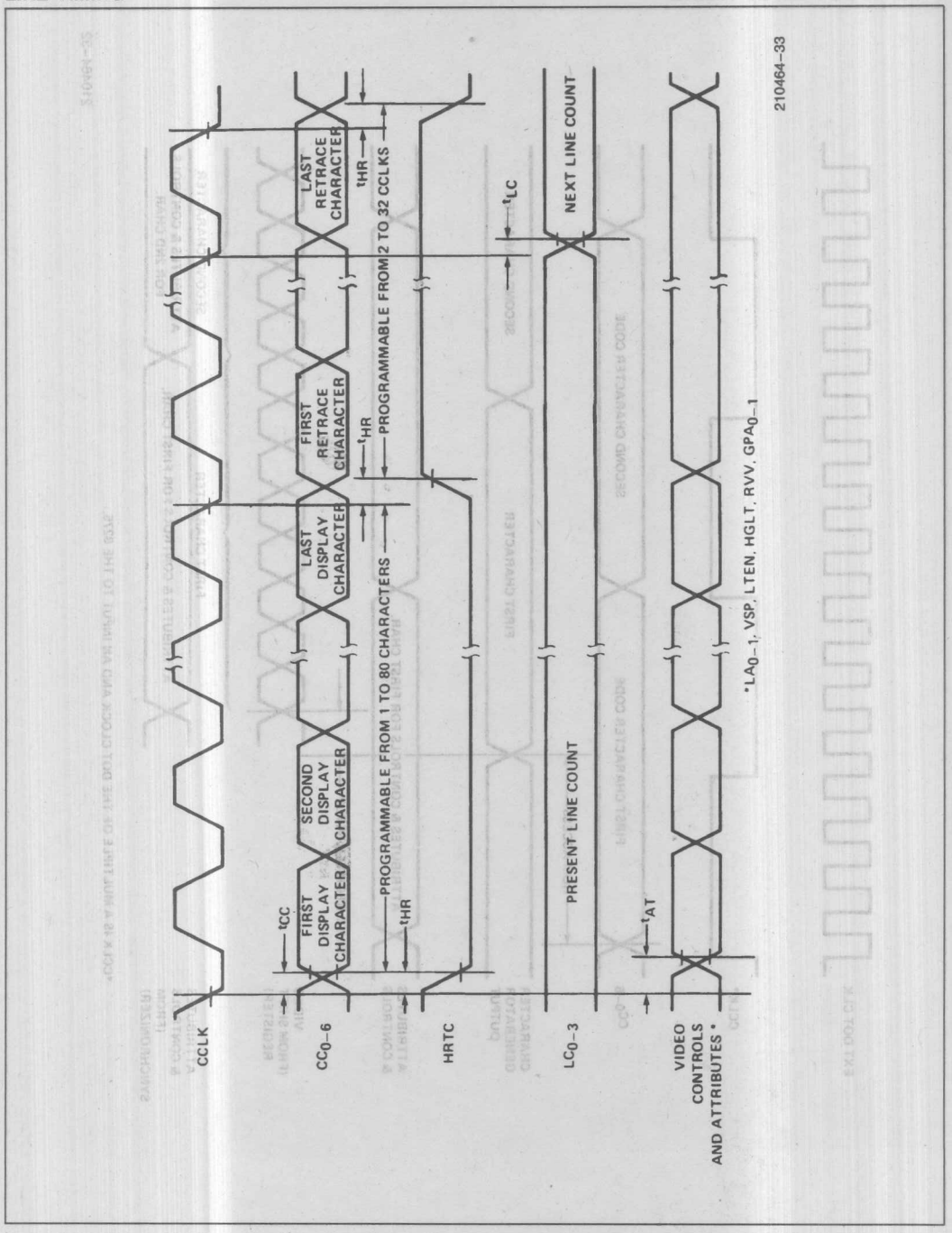
TYPICAL DOT LEVEL TIMING

*CCLK IS A MULTIPLE OF THE DOT CLOCK AND AN INPUT TO THE 8275.

210464-32

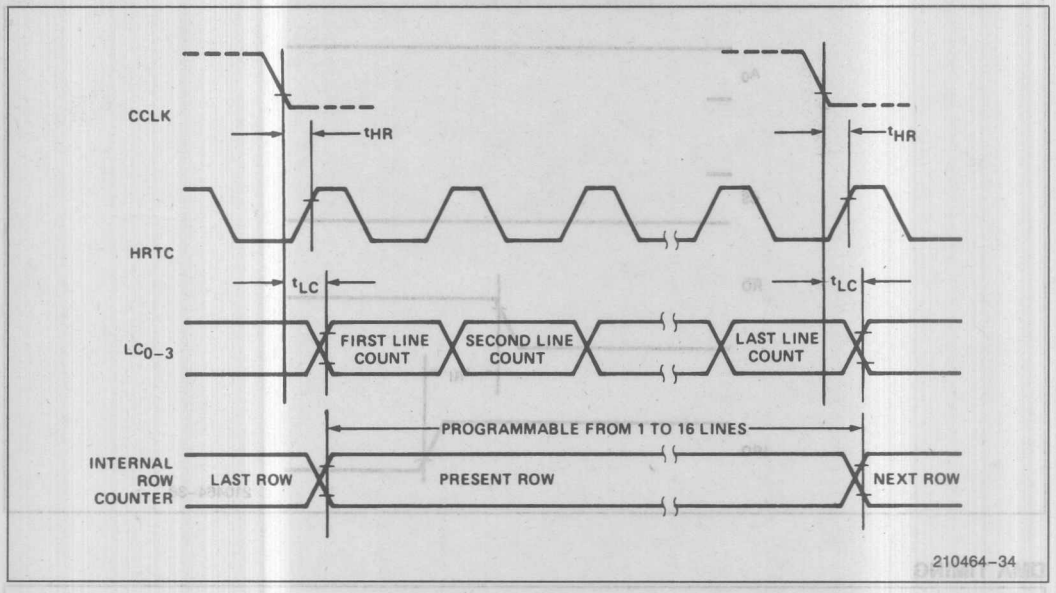
WAVEFORMS (Continued)

LINE TIMING

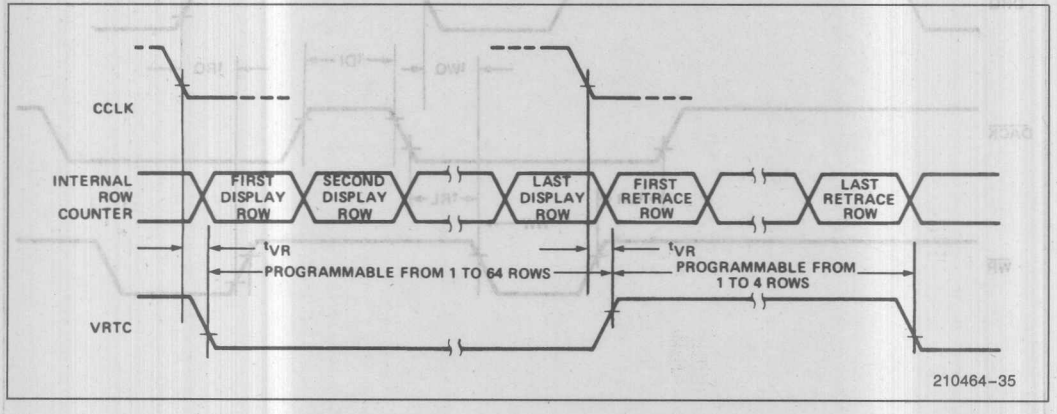


WAVEFORMS (Continued)

ROW TIMING

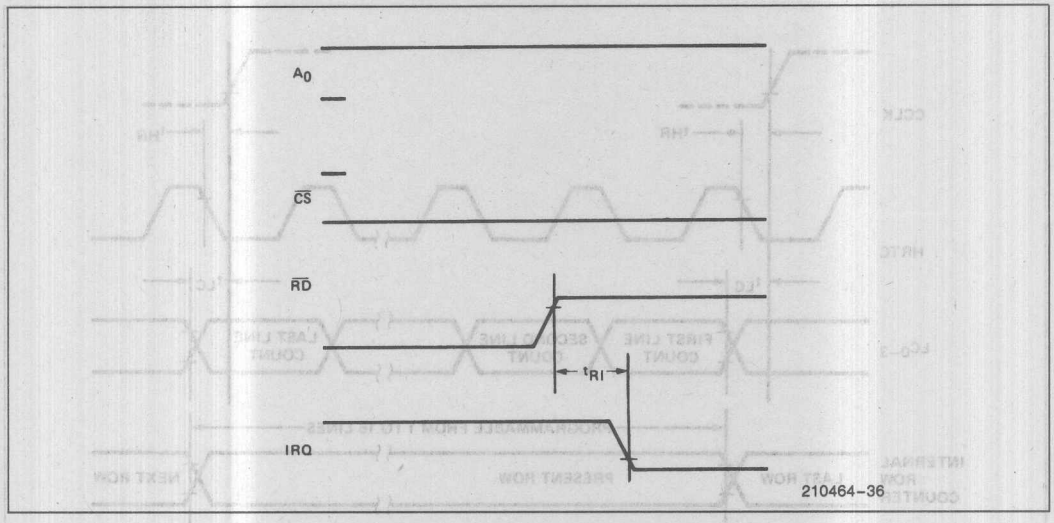


FRAME TIMING

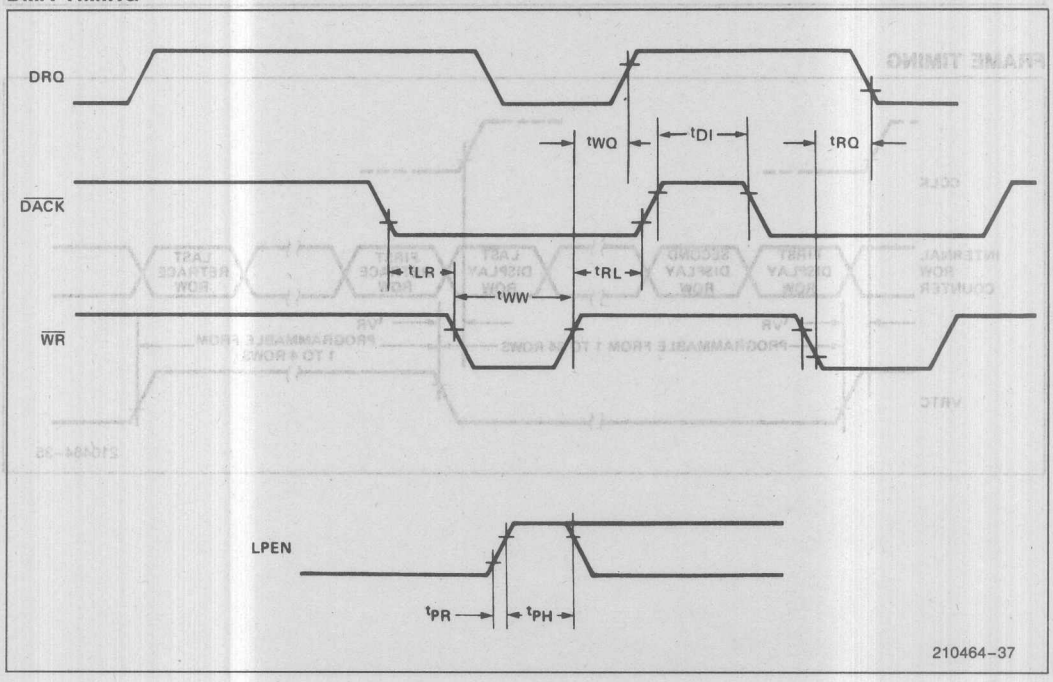


WAVEFORMS (Continued)

INTERRUPT TIMING

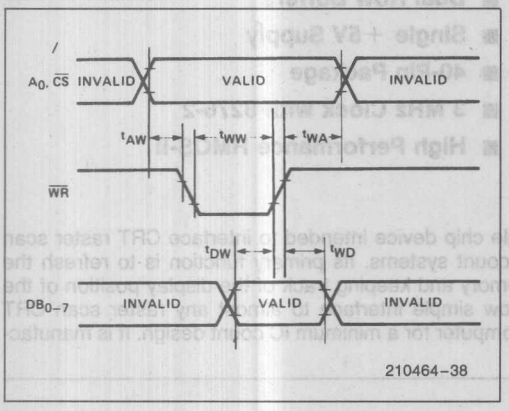


DMA TIMING

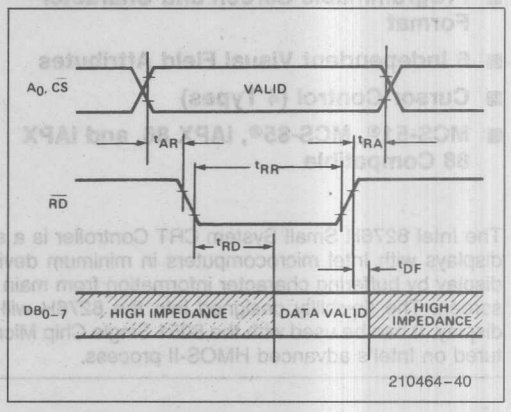


WAVEFORMS (Continued)

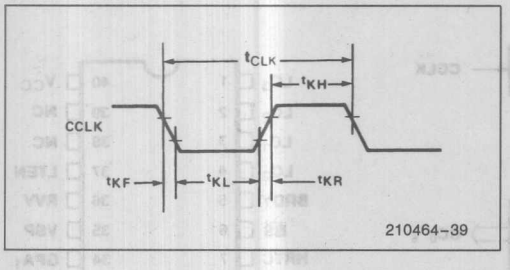
WRITE TIMING



READ TIMING



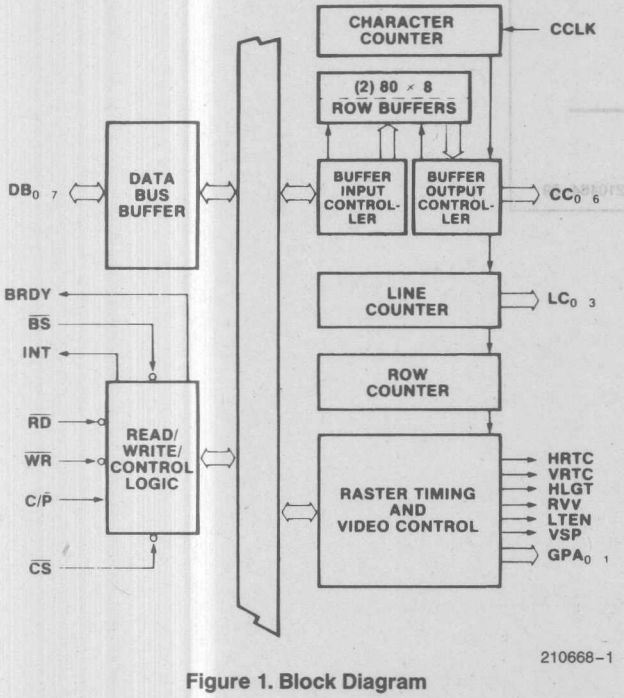
CLOCK TIMING



8276H
SMALL SYSTEM CRT CONTROLLER

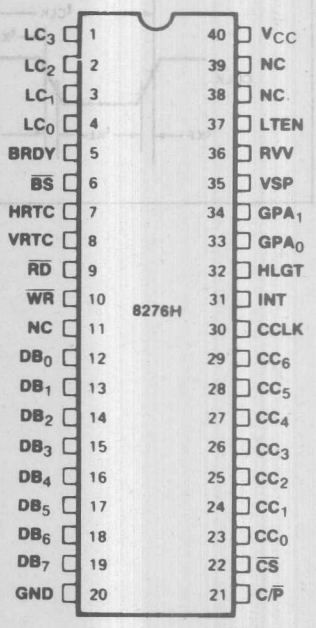
- Programmable Screen and Character Format
- 6 Independent Visual Field Attributes
- Cursor Control (4 Types)
- MCS-51®, MCS-85®, iAPX 86, and iAPX 88 Compatible
- Dual Row Buffers
- Single +5V Supply
- 40-Pin Package
- 3 MHz Clock with 8276-2
- High Performance HMOS-II

The Intel 8276H Small System CRT Controller is a single chip device intended to interface CRT raster scan displays with Intel microcomputers in minimum device-count systems. Its primary function is to refresh the display by buffering character information from main memory and keeping track of the display position of the screen. The flexibility designed into the 8276H will allow simple interface to almost any raster scan CRT display. It can be used with the 8051 Single Chip Microcomputer for a minimum IC count design. It is manufactured on Intel's advanced HMOS-II process.



210668-1

Figure 1. Block Diagram



210668-2

Figure 2. Pin configuration

November 1986
Order Number: 210668-002

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
LC_3 LC_2 LC_1 LC_0	1 2 3 4	O	**LINE COUNT:** Output from the line counter which is used to address the character generator for the line positions on the screen.
BRDY	5	O	**BUFFER READY:** Output signal indicating that a Row Buffer is ready for loading of character data.
$\overline{BS}$	6	I	**BUFFER SELECT:** Input signal enabling $\overline{WR}$ for character data into the Row Buffers.
HRTC	7	O	**HORIZONTAL RETRACE:** Output signal which is active during the programmed horizontal retrace interval. During this period the VSP output is high and the LTEN output is low.
VRTC	8	O	**VERTICAL RETRACE:** Output signal which is active during the programmed vertical retrace interval. During this period the VSP output is high and the LTEN output is low.
$\overline{RD}$	9	I	**READ INPUT:** A control signal to read registers.
$\overline{WR}$	10	I	**WRITE INPUT:** A control signal to write commands into the control registers or write data into the row buffers.
NC	11		**No Connection.**
DB_0 DB_1 DB_2 DB_3 DB_4 DB_5 DB_6 DB_7	12 13 14 15 16 17 18 19	I/O	**BIDIRECTIONAL DATA BUS:** Three-state lines. The outputs are enabled during a read of the C or P ports.
Ground	20		**Ground.**
V_{CC}	40		**+5V Power Supply.**
NC	39		**No Connection.**
NC	38		**No Connection.**
LTN	37	O	**LIGHT ENABLE:** Output signal used to enable the video signal to the CRT. This output is active at the programmed underline cursor position, and at positions specified by attribute codes.
RVV	36	O	**REVERSE VIDEO.** Output signal used to activate the CRT circuitry to reverse the video signal. This output is active at the cursor position if a reverse video block cursor is programmed or at the positions specified by the field attribute codes.
VSP	35	O	**VIDEO SUPPRESSION.** Output signal used to blank the video signal to the CRT. This output is active: —during the horizontal and vertical retrace intervals. —at the top and bottom lines of rows if underline is programmed to be number 8 or greater. —when an end of row or end of screen code is detected. —when a Row Buffer underrun occurs. —at regular intervals ($1/16$ frame frequency for cursor, $1/32$ frame frequency for attributes)—to create blinking displays as specified by cursor or field attribute programming.
GPA_1 GPA_0	34 33	O	**GENERAL PURPOSE ATTRIBUTE CODES:** Outputs which are enabled by the general purpose field attribute codes.
HLGT	32	O	**HIGHLIGHT:** Output signal used to intensify the display at particular positions on the screen as specified by the field atrribute codes.
INT	31	O	**INTERRUPT OUTPUT.**
CCLK	30	I	**CHARACTER CLOCK:** (from dot/timing logic).

Table 1. Pin Descriptions (Continued)

Symbol	Pin No.	Type	Name and Function
CC_6 CC_5 CC_4 CC_3 CC_2 CC_1 CC_0	29 28 27 26 25 24 23	O	**CHARACTER CODES:** Output from the row buffers used for character selection in the character generator.
$\overline{CS}$	22	I	**CHIP SELECT:** Enables $\overline{RD}$ of status or $\overline{WR}$ of command or parameters.
C/ P	21	I	**PORT ADDRESS:** A high input on this pin selects the "C" port or command registers and a low input selects the "P" port or parameter registers.

FUNCTIONAL DESCRIPTION

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8276H to the system Data Bus.

This functional block accepts inputs from the System Control Bus and generates control signals for overall device operation. It contains the Command, Parameter, and Status Registers that store the various control formats for the device functional definition.

C/$\overline{P}$	Operation	Register
0	Read	Reserved
0	Write	Parameter
1	Read	Status
1	Write	Command

$\overline{RD}$ (READ)

A "low" on this input informs the 8276H that the CPU is reading status information from the 8276H.

$\overline{WR}$ (WRITE)

A "low" on this input informs the 8276H that the CPU is writing data or control words to the 8276H.

$\overline{CS}$ (CHIP SELECT)

A "low" on this input selects the 8276H for $\overline{RD}$ or $\overline{WR}$ of Commands, Status, and Parameters.

BRDY (BUFFER READY)

A "high" on this output indicates that the 8276H is ready to receive character data.

$\overline{BS}$ (BUFFER SELECT)

A "low" on this input enables $\overline{WR}$ of character data to the 8276H row buffers.

INT (INTERRUPT)

A "high" on this output informs the CPU that the 8276H needs interrupt service.

C/$\overline{P}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	$\overline{BS}$	
0	0	1	0	1	Reserved
0	1	0	0	1	Write 8276H Parameter
1	0	1	0	1	Read 8276H Status
1	1	0	0	1	Write 8276H Command
X	1	0	1	0	Write 8276H Row Buffer
X	1	1	X	X	High Impedance
X	X	X	1	1	High Impedance

Character Counter

The Character Counter is a programmable counter that is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. It is driven by the CCLK (Character Clock) input, which should be derived from the external dot clock.

Line Counter

The Line Counter is a programmable counter that is used to determine the number of horizontal lines (Raster Scans) per character row. Its outputs are used to address the external character generator.

Row Counter

The Row Counter is a programmable counter that is used to determine the number of character rows to be displayed per frame and length of the vertical retrace interval.

Raster Timing and Video Controls

The Raster Timing circuitry controls the timing of the HRTC (Horizontal Retrace) and VRTC (Vertical Retrace) outputs. The Video control circuitry controls the generation of HGLT (Highlight), RVV (Reverse Video), LTEN (Light Enable), VSP (Video Suppress), and GPA_{0-1} (General Purpose Attribute) outputs.

Row Buffers

The Row Buffers are two 80-character buffers. They are filled from the microcomputer system memory with the character codes to be displayed. While one row buffer is displaying a row of characters, the other is being filled with the next row of characters.

Buffer Input/Output Controllers

The Buffer Input/Output Controllers decode the characters being placed in the row buffers. If the character is a field attribute or special code, they control the appropriate action. (Example: A "Highlight" field atrribute will cause the Buffer Output Controller to activate the HGLT output.)

SYSTEM OPERATION

The 8276H is programmable to a large number of different display formats. It provides raster timing, display row buffering, visual attribute decoding and cursor timing.

It is designed to interface with standard character generators for dot matrix decoding. Dot level timing must be provided by external circuitry.

General Systems Operational Description

Display characters are retrieved from memory and displayed on a row-by-row basis. The 8276H has two row buffers. While one row buffer is being used for display, the other is being filled with the next row of characters to be displayed. The number of display characters per row and the number of character rows per frame are software programmable, providing easy interface to most CRT displays. (See Programming Section.)

The 8276H uses BRDY to request character data to fill the row buffer that is not being used for display.

The 8276H displays character rows one scan line at a time. The number of scan lines per character row, the underline position, and blanking of top and bottom lines are programmable. (See Programming Section.)

The 8276H provides special Control Codes which can be used to minimize overhead. It also provides

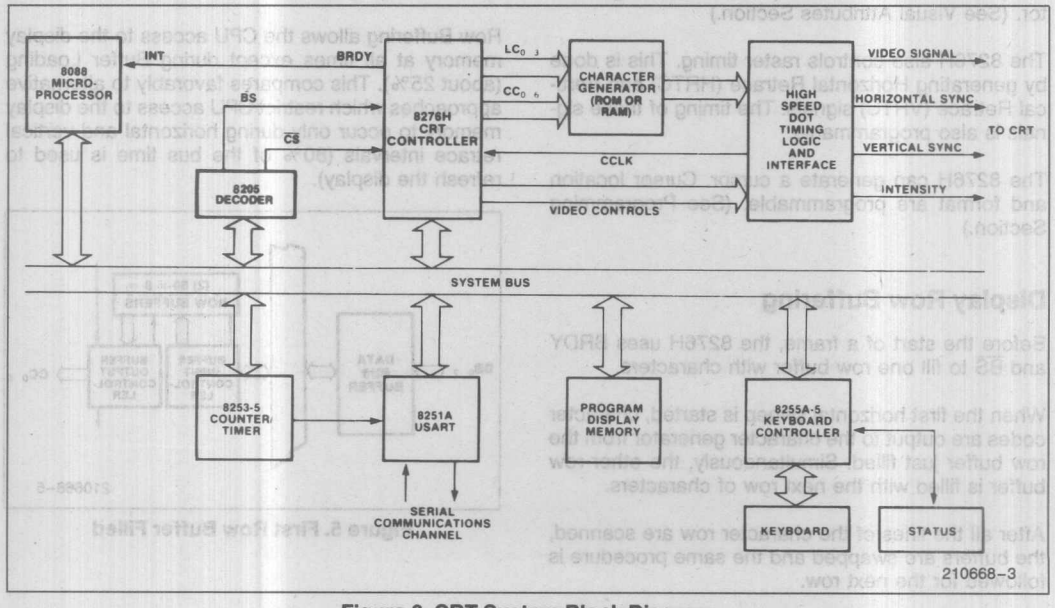


Figure 3. CRT System Block Diagram

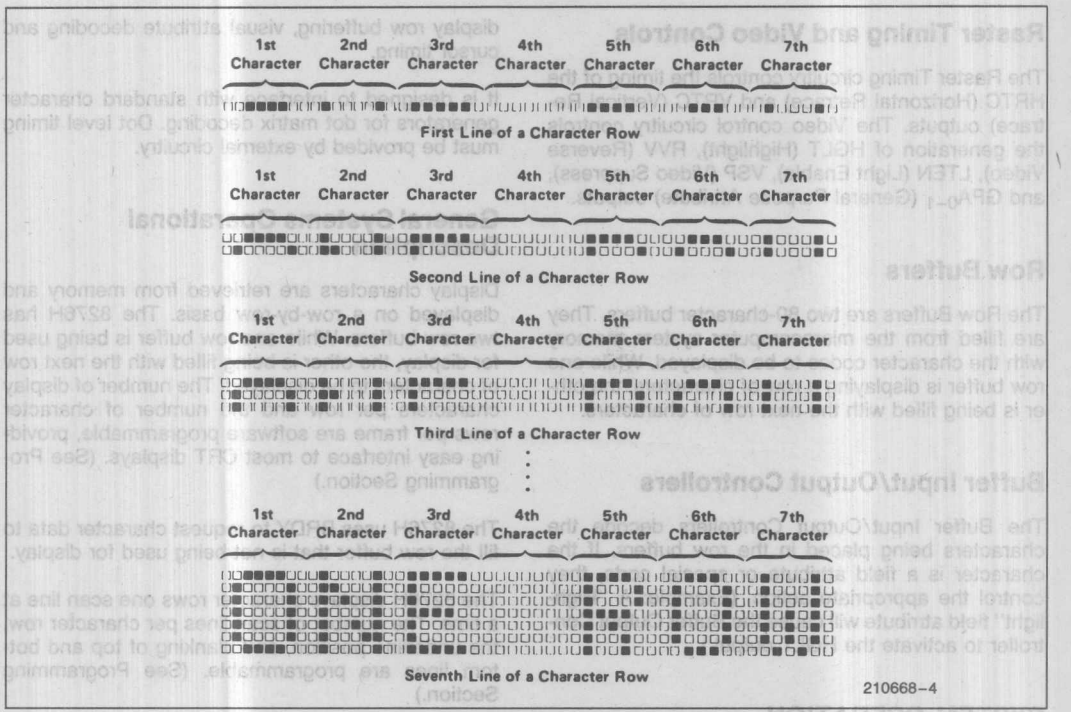


Figure 4. Display Of A Character Row

Visual Attribute Codes to cause special action on the screen without the use of the character generator. (See Visual Attributes Section.)

The 8276H also controls raster timing. This is done by generating Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) signals. The timing of these signals is also programmable.

The 8276H can generate a cursor. Cursor location and format are programmable. (See Programming Section.)

Display Row Buffering

Before the start of a frame, the 8276H uses BRDY and $\overline{BS}$ to fill one row buffer with characters.

When the first horizontal sweep is started, character codes are output to the character generator from the row buffer just filled. Simultaneously, the other row buffer is filled with the next row of characters.

After all the lines of the character row are scanned, the buffers are swapped and the same procedure is followed for the next row.

This process is repeated until all of the character rows are displayed.

Row Buffering allows the CPU access to the display memory at all times except during Buffer Loading (about 25%). This compares favorably to alternative approaches which restrict CPU access to the display memory to occur only during horizontal and vertical retrace intervals (80% of the bus time is used to refresh the display).

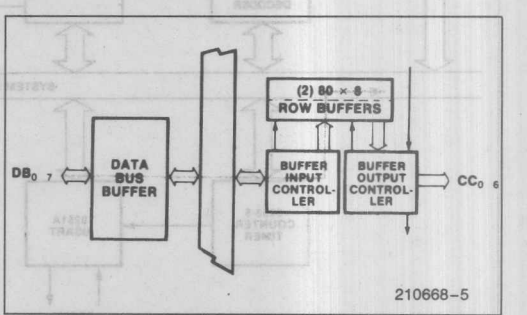


Figure 5. First Row Buffer Filled

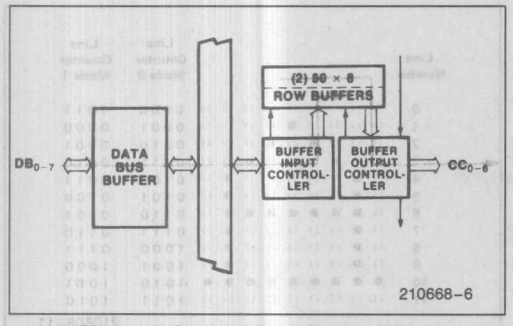


Figure 6. Second Row Buffer Filled, First Row Displayed

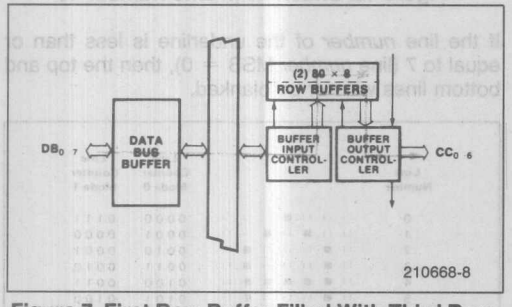


Figure 7. First Row Buffer Filled With Third Row, Second Row Displayed

Display Format

SCREEN FORMAT

The 8276H can be programmed to generate from 1 to 80 characters per row, and from 1 to 64 rows per frame.

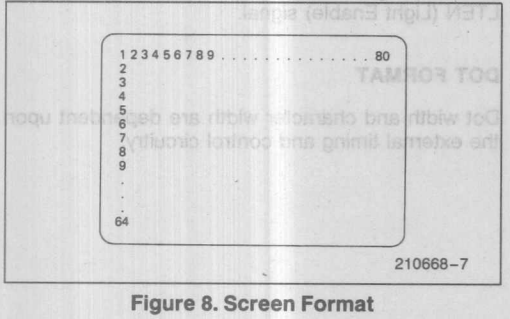


Figure 8. Screen Format

The 8276H can also be programmed to blank alternate rows. In this mode, the first row is displayed, the second blanked, the third displayed, etc. Display data is not requested for the blanked rows.

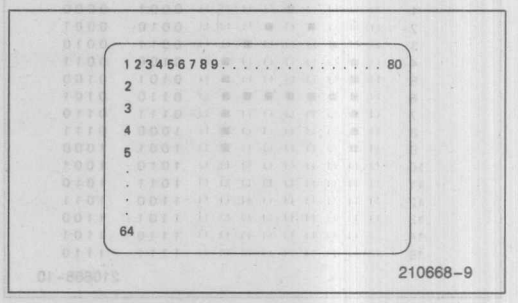


Figure 9. Blank Alternate Rows Mode

ROW FORMAT

The 8276H is designed to hold the line count stable while outputting the appropriate character codes during each horizontal sweep. The line count is incremented during horizontal retrace and the whole row of character codes are output again during the next sweep. This is continued until the entire character row is displayed.

The number of lines (horizontal sweeps) per character row is programmable from 1 to 16.

The output of the line counter can be programmed to be in one of two modes.

In mode 0, the output of the line *counter* is the same as the line *number.*

In mode 1, the line *counter* is offset by one from the line *number.*

NOTE:
In mode 1, while the *first* line (line number 0) is being displayed, the *last* count is output by the line counter (see examples).

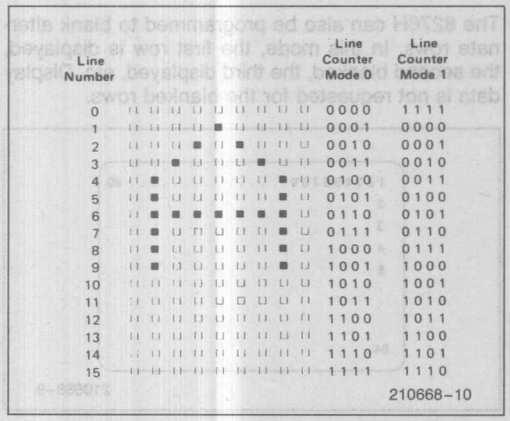


Figure 10. Example of a 16-Line Format

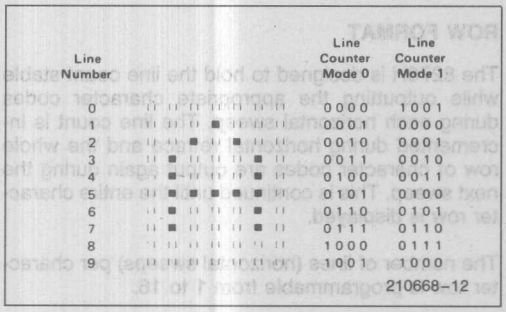


Figure 11. Example of a 10-Line Format

Mode 0 is useful for character generators that leave address zero blank and start at address 1. Mode 1 is useful for character generators which start at address zero.

Underline placement is also programmable (from line *number* 0 to 15). This is independent of the line *counter* mode.

If the line *number* of the underline is greater than 7 (line *number* MSB = 1), then the top and bottom lines will be blanked.

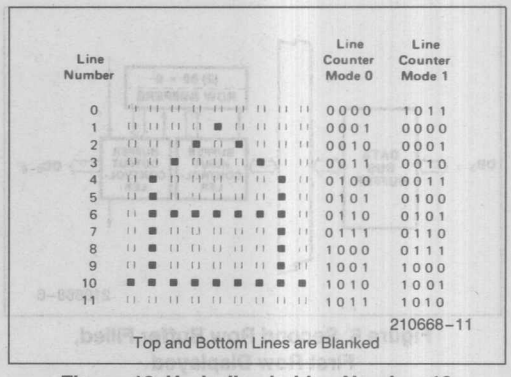


Figure 12. Underline in Line Number 10

If the line *number* of the underline is less than or equal to 7 (line *number* MSB = 0), then the top and bottom lines will *not* be blanked.

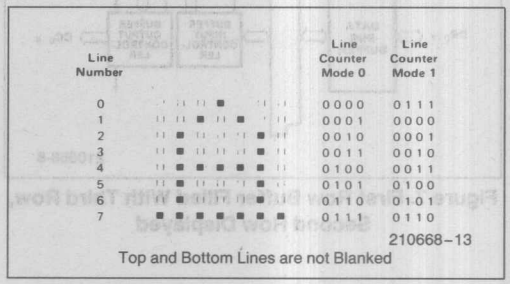


Figure 13. Underline in Line Number 7

If the line *number* of the underline is greater than the maximum number of lines, the underline will not appear.

Blanking is accomplished by the VSP (Video Suppression) signal. Underline is accomplished by the LTEN (Light Enable) signal.

DOT FORMAT

Dot width and character width are dependent upon the external timing and control circuitry.

Dot level timing circuitry should be designed to accept the parallel output of the character generator and shift it out serially at the rate required by the CRT display.

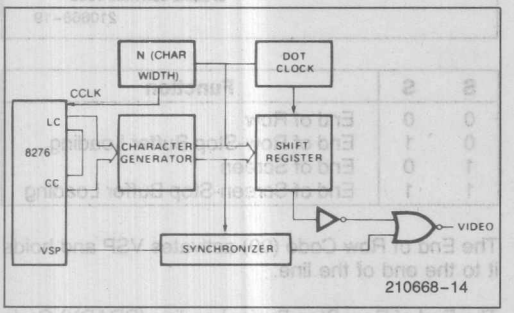


Figure 14. Typical Dot Level Block Diagram

Dot width is a function of dot clock frequency.

Character width is a function of the character generator width.

Horizontal character spacing is a function of the shift register length.

NOTE:
Video control and timing signals must be synchronized with the video signal due to the character generator access delay.

Raster Timing

The character counter is driven by the character clock input (CCLK). It counts out the characters being displayed (programmable from 1 to 80). It then causes the line counter to increment, and it starts counting out the horizontal retrace interval (programmable from 2 to 32). This process is constantly repeated.

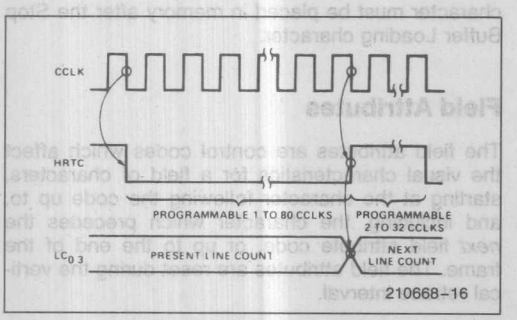


Figure 15. Line Timing

The line counter is driven by the character counter. It is used to generate the line address outputs (LC_{0-3}) for the character generator. After it counts all of the lines in a character row (programmable from 1 to 16), it increments the row counter, and starts over again. (See Character Format Section for detailed description of Line Counter functions.)

The row counter is an internal counter driven by the line counter. It controls the functions of the row buffers and counts the number of character rows displayed.

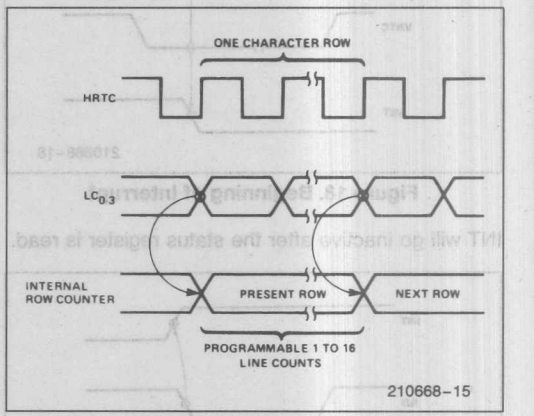


Figure 16. Row Timing

After the row counter counts all of the rows in a frame (programmable from 1 to 64), it starts counting out the vertical retrace interval (programmable from 1 to 4).

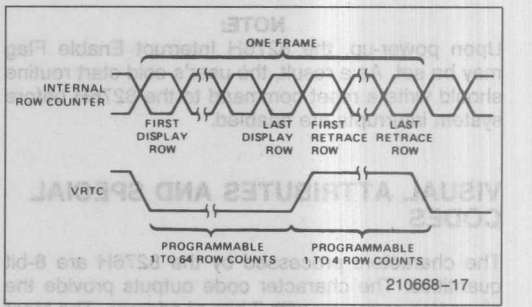


Figure 17. Frame Timing

The Video Suppression Output (VSP) is active during horizontal and vertical retrace intervals.

Dot level timing circurity must synchronize these outputs with the video signal to the CRT Display.

Interrupt Timing

The 8276H can be programmed to generate an interrupt request at the end of each frame. If the 8276H interrupt enable flag is set, an interrupt request will occur at the *beginning* of the *last display row.*

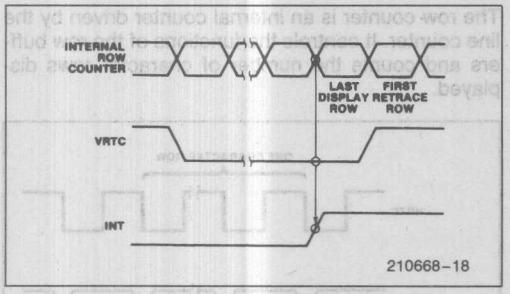


Figure 18. Beginning of Interrupt

INT will go inactive after the status register is read.

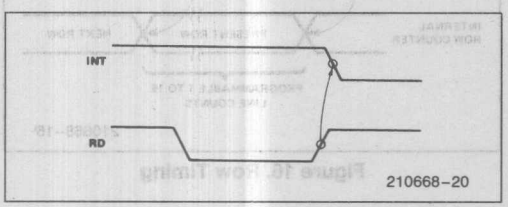


Figure 19. End of Interrupt

A reset command will also cause INT to go inactive, but this is not recommended during normal service.

NOTE:
Upon power-up, the 8276H Interrupt Enable Flag may be set. As a result, the user's cold start routine should write a reset command to the 8276H before system interrupts are enabled.

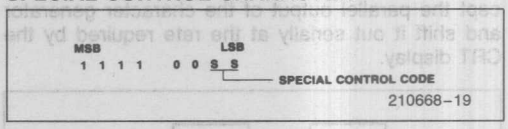
VISUAL ATTRIBUTES AND SPECIAL CODES

The characters processed by the 8276H are 8-bit quantities. The character code outputs provide the character generator with 7 bits of address. The Most Significant Bit is the extra bit and it is used to determine if it is a normal display character (MSB = 0), or if it is a Field Attribute or Special Code (MSB = 1).

Special Codes

Four special codes are available to help reduce bus usage.

SPECIAL CONTROL CHARACTER



S	S	Function
0	0	End of Row
0	1	End of Row-Stop Buffer Loading
1	0	End of Screen
1	1	End of Screen-Stop Buffer Loading

The End of Row Code (00) activates VSP and holds it to the end of the line.

The End of Row-Stop Buffer Loading (BRADY) Code (01) causes the Buffer Loading Control Logic to stop buffer loading for the rest of the row upon being written into the Row Buffer. It affects the display in the same way as the End of Row Code (00).

The End of Screen Code (10) activates VSP and holds it to the end of the frame.

The End of Screen-Stop Buffer Loading (BRDY) Code (11) causes the Row Buffer Control Logic to stop buffer loading for the rest of the frame upon being written. It affects the display in the same way as the End of Screen Code (10).

If the Stop Buffer Loading feature is not used, all characters after an End of Row character are ignored, except for the End of Screen character, which operates normally. All characters after an End of Screen character, are ignored.

NOTE:
If a Stop Buffer Loading is not the last character in a row, Buffer Loading is not stopped until after the next character is read. In this situation, a dummy character must be placed in memory after the Stop Buffer Loading character.

Field Attributes

The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the code up to, and including, the character which precedes the *next* field attribute code, or up to the end of the frame. The field attributes are reset during the vertical retrace interval.

The 8276H can be programmed to provide visible field attribute characters; all field attribute codes will occupy a position on the screen. These codes will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character.

There are six field attributes:

1) *Blink*—Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2) *Highlight*—Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).
3) *Reverse Video*—Characters following the code are caused to appear with reverse video by activating the Reverse Video output (RVV).
4) *Underline*—Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).
5, 6) *General Purpose*—There are two additional 8276H outputs which act as general purpose, independently programmable field attributes. GPA_{0-1} are active high outputs.

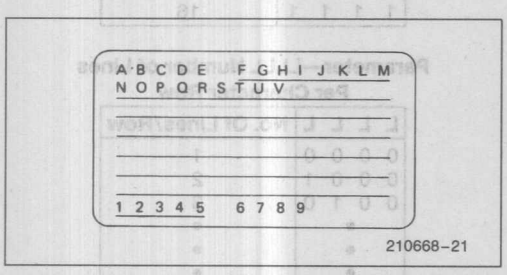
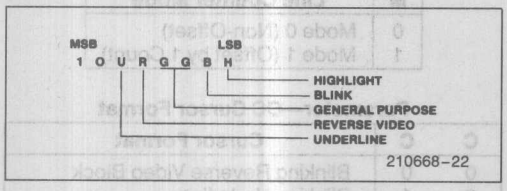


Figure 20. Example of a Visible Field Attribute (Underline Attribute)

FIELD ATTRIBUTE CODE



H = 1 FOR HIGHLIGHTING
B = 1 FOR BLINKING
R = 1 FOR REVERSE VIDEO
U = 1 FOR UNDERLINE
GG = 1 FOR GPA_1, GPA_0

NOTE:
More than one attribute can be enabled at the same time. If the blinking and reverse video attributes are enabled simultaneously, only the reversed characters will blink.

Cursor Timing

The cursor location is determined by a cursor row register and a character position register which are loaded by command to the controller. The cursor can be programmed to appear on the display as:

1. a blinking underline
2. a blinking reverse video block
3. a non-blinking underline
4. a non-blinking reverse video block

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

If a non-blinking reverse video *cursor* appears in a non-blinking reverse video *field,* the cursor will appear as a normal video block.

If a non-blinking underline *cursor* appears in a non-blinking underline *field,* the cursor will not be visible.

Device Programming

The 8276H has two programming registers, the Command Register and the Parameter Register. It also has a Status Register. The Command Register can only be written into and the Status Register can only be read from. They are addressed as follows:

C/$\overline{P}$	Operation	Register
0	Read	Reserved
0	Write	Parameter
1	Read	Status
1	Write	Command

The 8276H expects to receive a command and a sequence of 0 to 4 parameters, depending on the command. If the proper number of parameter bytes are not received before another command is given, a status flag is set, indicating an improper command.

Instruction Set

The 8276H instruction set consists of 7 commands

Command	No. of Parameter Bytes
Reset	4
Start Display	0
Stop Display	0
Load Cursor	2
Enable Interrupt	0
Disable Interrupt	0
Preset Counters	0

In addition, the status of the 8276H can be read by the CPU at any time.

1. RESET COMMAND

	Operation	C/P	Description	Data Bus MSB LSB
Command	Write	1	Reset Command	0 0 0 0 0 0 0 0
Parameters	Write	0	Screen Comp Byte 1	S H H H H H H H
	Write	0	Screen Comp Byte 2	V V R R R R R R
	Write	0	Screen Comp Byte 3	U U U U L L L L
	Write	0	Screen Comp Byte 4	M 1 C C Z Z Z Z

Action—After the reset command is written, BRDY goes inactive, 8276H interrupts are disabled, and the VSP output is used to blank the screen. HRTC and VRTC continue to run. HRTC and VRTC timing are random on power-up.

As parameters are written, the screen composition is defined.

Parameter—S Spaced Rows

S	Functions
0	Normal Rows
1	Spaced Rows

Parameter—HHHHHHH Horizontal Characters/Row

H	H	H	H	H	H	H	No. Of Characters Per Row
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	2
0	0	0	0	0	1	0	3
			•				•
			•				•
			•				•
1	0	0	1	1	1	1	80
1	0	1	0	0	0	0	Undefined
			•				•
			•				•
			•				•
1	1	1	1	1	1	1	Undefined

Parameter—VV Vertical Retrace Row Count

V	V	No. Of Row Counts Per VRTC
0	0	1
0	1	2
1	0	3
1	1	4

Parameter—RRRRRR Vertical Rows/Frame

R	R	R	R	R	R	No. Of Rows/Frame
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
			•			•
			•			•
			•			•
1	1	1	1	1	1	64

Parameter—UUUU Underline Placement

U	U	U	U	Line Number Of Underline
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
		•		•
		•		•
		•		•
1	1	1	1	16

Parameter—LLLL Number of Lines Per Character Row

L	L	L	L	No. Of Lines/Row
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
		•		•
		•		•
		•		•
1	1	1	1	16

Parameter—M Line Counter Mode

M	Line Counter Mode
0	Mode 0 (Non-Offset)
1	Mode 1 (Offset by 1 Count)

Parameter—CC Cursor Format

C	C	Cursor Format
0	0	Blinking Reverse Video Block
0	1	Blinking Underline
1	0	Non-blinking Reverse Video Block
1	1	Non-blinking Underline

Parameter—ZZZZ Horizontal Retrace Count

Z Z Z Z	No. Of Character Counts Per HRTC
0 0 0 0	2
0 0 0 1	4
0 0 1 0	6
•	•
•	•
•	•
1 1 1 1	32

NOTE:
uuuu MSB determines blanking of top and bottom lines (1 = blanked, 0 = not blanked).

2. START DISPLAY COMMAND

	Operation	C/P	Description	Data Bus MSB LSB
Command	Write	1	Start Display	0 0 1 0 0 0 0 0
No Parameters				

Action—8276H interrupts are enabled, BRDY goes active, video is enabled, Interrupt Enable and Video Enable status flags are set.

3. STOP DISPLAY COMMAND

	Operation	C/P	Description	Data Bus MSB LSB
Command	Write	1	Stop Display	0 1 0 0 0 0 0 0
No Parameters				

Action—Disables video, interrupts remain enabled, HRTC and VRTC continue to run, Video Enable status flag is reset, and the "Start Display" command must be given to reenable the display.

4. LOAD CURSOR POSITION

	Operation	C/P	Description	Data Bus MSB LSB
Command	Write	1	Load Cursor	1 0 0 0 0 0 0 0
Parameters	Write	0	Char. Number	(Char. Position in Row)
	Write	0	Row Number	(Row Number)

Action—The 8276H is conditioned to place the next two parameter bytes into the cursor position registers. Status flag not affected.

5. ENABLE INTERRUPT COMMAND

	Operation	C/P	Description	Data Bus MSB LSB
Command	Write	1	Enable Interrupt	1 0 1 0 0 0 0 0
No Parameters				

Action—The interrupt enable flag is set and interrupts are enabled.

6. DISABLE INTERRUPT COMMAND

	Operation	C/P	Description	Data Bus MSB LSB
Command	Write	1	Disable Interrupt	1 0 1 0 0 0 0 0
No Parameters				

Action—Interrupts are disabled and the interrupt enable status flag is reset.

7. PRESET COUNTERS COMMAND

	Operation	C/P	Description	Data Bus MSB LSB
Command	Write	1	Preset Counters	1 1 1 0 0 0 0 0
No Parameters				

Action—The internal timing counters are preset, corresponding to a screen display position at the top left corner. Two character clocks are required for this operation. The counters will remain in this state until any other command is given.

This command is useful for system debug and synchronization of clustered CRT displays on a single CPU. After this command, two additional clock cycles are required before the first character of the first row is put out.

Status Flags

	Operation	C/P	Description	Data Bus MSB LSB
Command	Read	1	Status Word	0 IE IR X IC VE BU X

IE — (Interrupt Enable) Set or reset by command. It enables vertical retrace interrupt. It is automatically set by a "Start Display" command and reset with the "Reset" command.

IR — (Interrupt Request) This flag is set at the beginning of display of the last row of the frame if the interrupt enable flag is set. It is reset after a status read operation.

IC — (Improper Command) This flag is set when a command parameter string is too long or too short. The flag is automatically reset after a status read.

VE — (Video Enable) This flag indicates that video operation of the CRT is enabled. This flag is set on a "Start Display" command, and reset on a "Stop Display" or "Reset" command.

BU — (Buffer Underrun) This flag is set whenever a Row Buffer is not filled with character data in time for a buffer swap required by the display. Upon activation of this bit, buffer loading ceases, and the screen is blanked until after vertical retrace interval.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On Any Pin
With Respect to Ground .....−0.5V to +7V
Power Dissipation .....1 Watt

**Nctice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

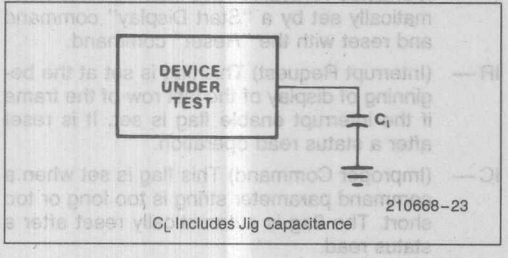
D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Ouput Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	

CAPACITANCE T_A = 25°C; V_{CC} = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_C = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}

A.C. TESTING LOAD CIRCUIT



C_L Includes Jig Capacitance

A.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V

Bus Parameters

READ CYCLE

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{AR}	Address Stable Before READ	0		ns	
t_{RA}	Address Hold Time for READ	0		ns	
t_{RR}	READ Pulse Width	250		ns	
t_{RD}	Data Delay from READ		200	ns	C_L = 150 pF
t_{DF}	READ to Data Floating		100	ns	

WRITE CYCLE

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{AW}	Address Stable Before WRITE	0		ns	
t_{WA}	Address Hold Time for WRITE	0		ns	
t_{WW}	WRITE Pulse Width	250		ns	
t_{DW}	Data Setup Time for WRITE	150		ns	
t_{WD}	Data Hold Time for WRITE	0		ns	

CLOCK TIMING

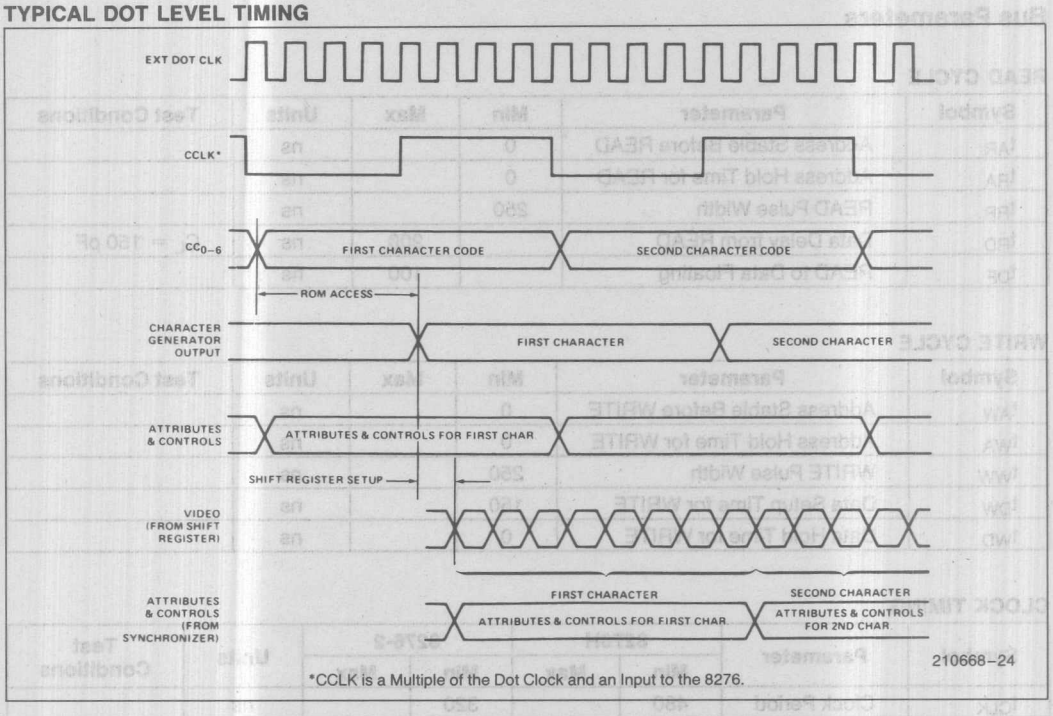
Symbol	Parameter	8276H		8276-2		Units	Test Conditions
		Min	Max	Min	Max		
t_{CLK}	Clock Period	480		320		ns	
t_{KH}	Clock High	240		120		ns	
t_{KL}	Clock Low	160		120		ns	
t_{KR}	Clock Rise	5	30	5	30	ns	
t_{KF}	Clock Fall	5	30	5	30	ns	

OTHER TIMING

Symbol	Parameter	8276H		8276-2		Units	Test Conditions
		Min	Max	Min	Max		
t_{CC}	Character Code Output Delay		150		150	ns	CL = 50 pF
t_{HR}	Horizontal Retrace Output Delay		200		150	ns	CL = 50 pF
t_{LC}	Line Count Output Delay		400		250	ns	CL = 50 pF
t_{AT}	Control/Attribute Output Delay		275		250	ns	CL = 50 pF
t_{VR}	Vertical Retrace Output Delay		275		250	ns	CL = 50 pF
t_{RI}	INT ↓ from RD ↑		250		250	ns	CL = 50 pF
t_{WQ}	BRDY ↑ from $\overline{WR}$ ↑		250		250	ns	CL = 50 pF
t_{RQ}	BRDY ↓ from $\overline{WR}$ ↓		200		200	ns	CL = 50 pF
t_{LR}	$\overline{BS}$ ↓ to $\overline{WR}$ ↓	0		0		ns	
t_{RL}	$\overline{WR}$ ↑ to BS ↑	0		0		ns	

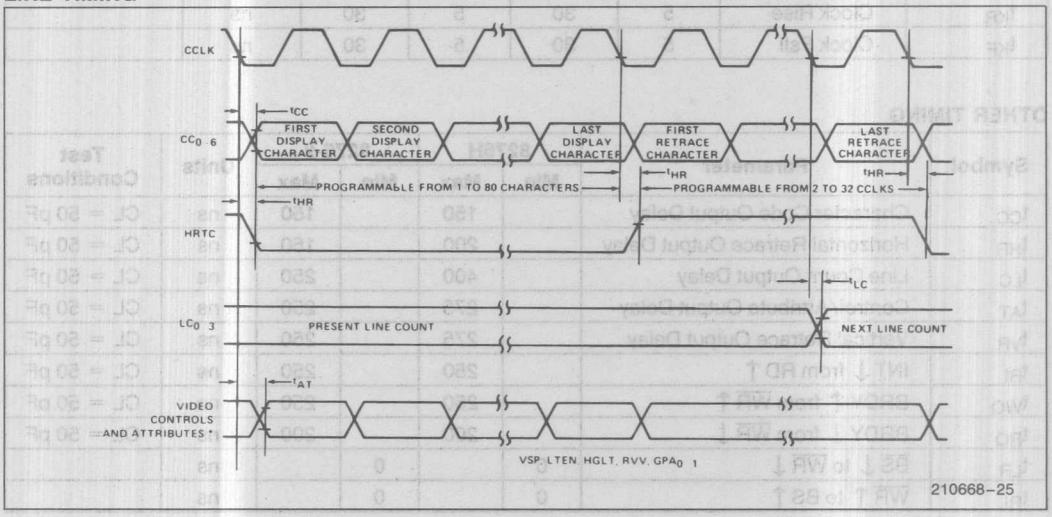
WAVEFORMS

TYPICAL DOT LEVEL TIMING

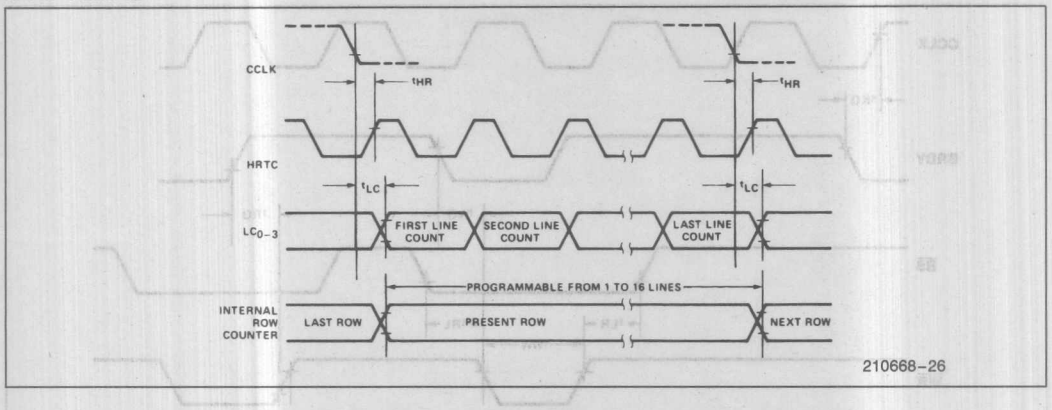


*CCLK is a Multiple of the Dot Clock and an Input to the 8276.

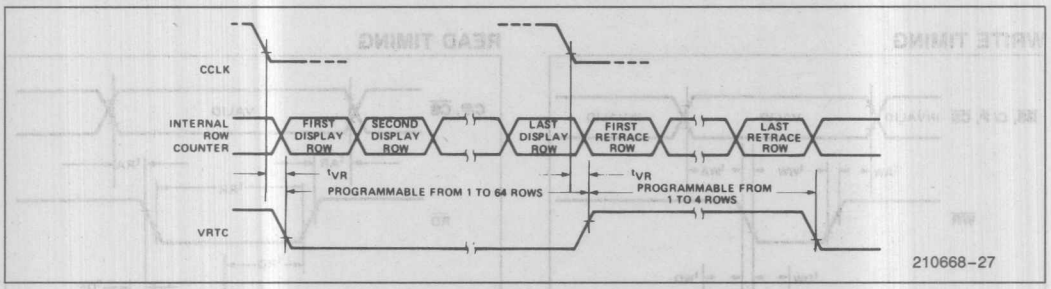
LINE TIMING



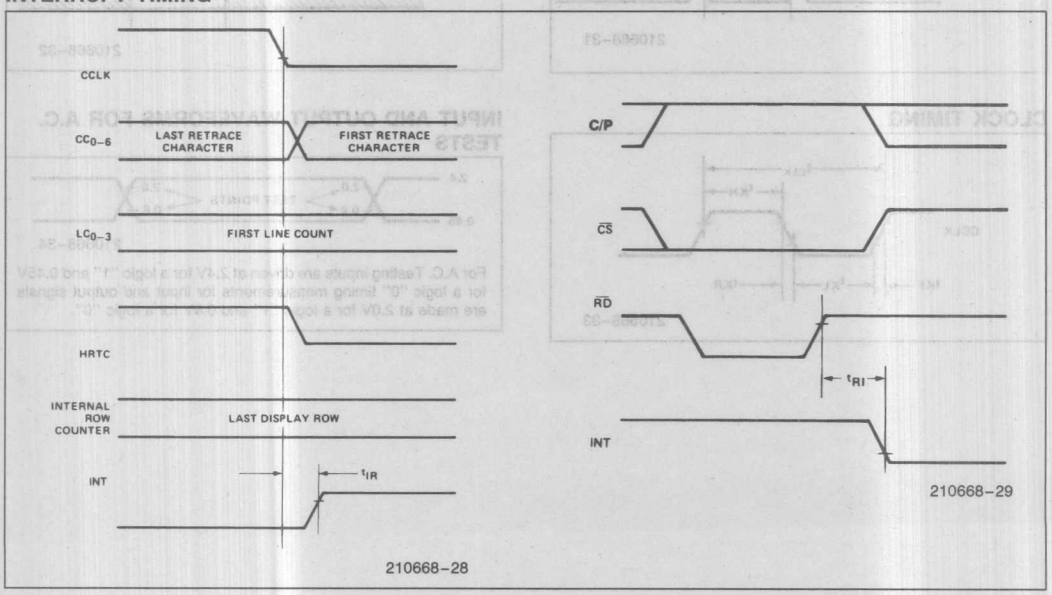
ROW TIMING



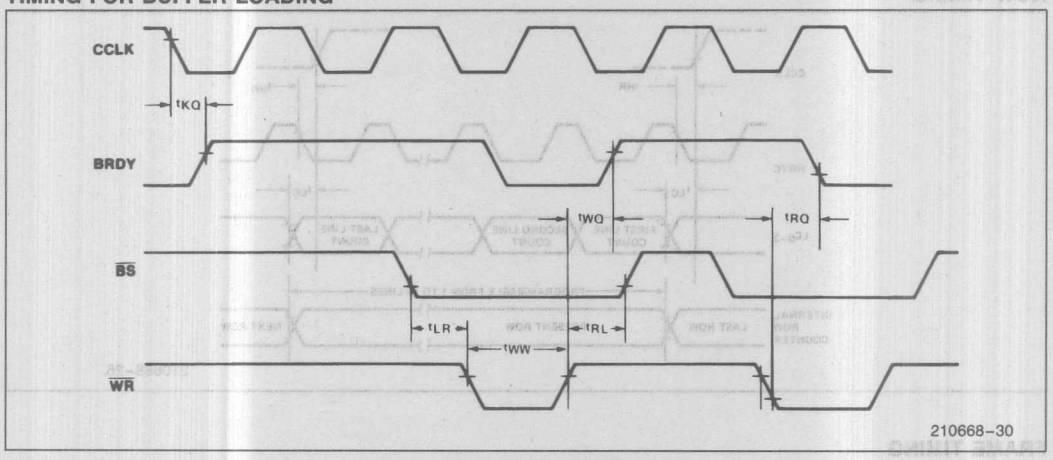
FRAME TIMING



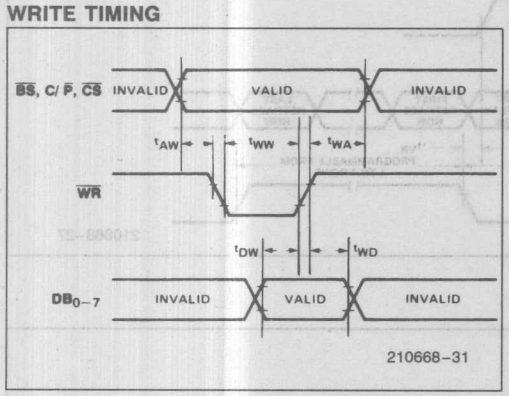
INTERRUPT TIMING



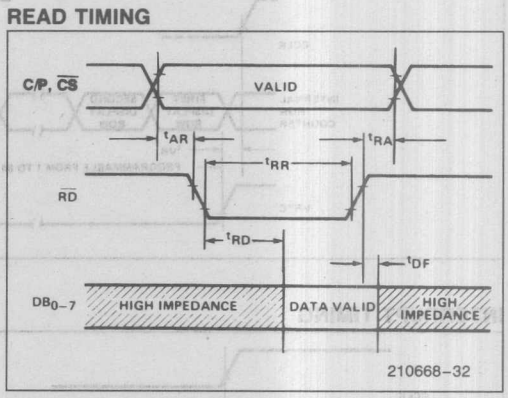
TIMING FOR BUFFER LOADING



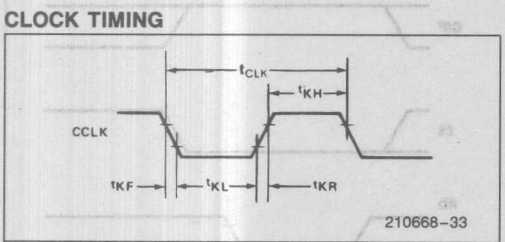
WRITE TIMING



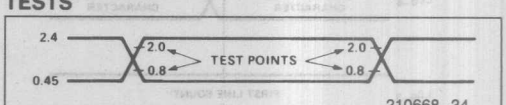
READ TIMING



CLOCK TIMING



INPUT AND OUTPUT WAVEFORMS FOR A.C. TESTS



For A.C. Testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0" timing measurements for input and output signals are made at 2.0V for a logic "1" and 0.8V for a logic "0".

intel

APPLICATION NOTE

AP-62

November 1986

A Low Cost CRT Terminal Using the 8275

JOHN KATAUSKY
PERIPHERALS APPLICATIONS

Order Number: 207780-001

intel

APPLICATION
NOTE

November 1988

A Low Cost CRT Terminal Using the 8275

JOHN KATAUSKY
PERIPHERALS APPLICATIONS

Order Number: 207780-001

1.0 INTRODUCTION

The purpose of this application note is to provide the reader with the design concepts and factual tools needed to integrate Intel peripherals and microprocessors into a low cost raster scan CRT terminal. A previously published application note, AP-32, presented one possible solution to the CRT design question. This application note expands upon the theme established in AP-32 and demonstrates how to design a functional CRT terminal while keeping the parts count to a minimum.

For convenience, this application note is divided into seven general sections:

1. Introduction
2. CRT Basics
3. 8275 Description
4. Design Background
5. Circuit Description
6. Software Description
7. Appendix

There is no question that microprocessors and LSI peripherals have had a significant role in the evolution of CRT terminals. Microprocessors have allowed design engineers to incorporate an abundance of sophisticated features into terminals that were previously mere slaves to a larger processor. To complement microprocessors, LSI peripherals have reduced component count in many support areas. A typical LSI peripheral easily replaces between 30 and 70 SSI and MSI packages, and offers features and flexibility that are usually not available in most hardware designs. In addition to replacing a whole circuit board of random logic, LSI circuits also reduce the cost and increase the reliability of design. Fewer interconnects increases mechanical reliability and fewer parts decreases the power consumption and hence, the overall reliability of the design. The reduction of components also yields a circuit that is easier to debug during the actual manufacturing phase of a product.

Until the era of advanced LSI circuitry, a typical CRT terminal consisted of 80 to 200 or more SSI and MSI packages. The first microprocessors and peripherals dropped this component count to between 30 and 50 packages. This application note describes a CRT terminal that uses 20 packages.

2.0 CRT BASICS

The raster scan display gets its name from the fact that the image displayed on the CRT is built up by generating a series of lines (raster) across the face of the CRT. Usually, the beam starts in the upper left hand corner of the display and simultaneously moves left to right and top to bottom to put a series of zig-zag lines on the screen (Figure 2.1). Two simultaneously operating independent circuits control the vertical and horizontal movement of the beam.

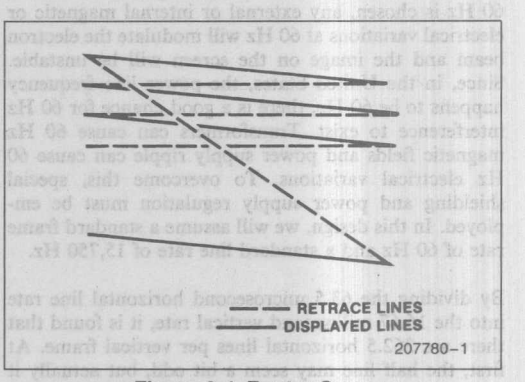


Figure 2-1. Raster Scan

As the electron beam moves across the face of the CRT, a third circuit controls the current flowing in the beam. By varying the current in the electron beam the image on the CRT can be made to be as bright or as dark as the user desires. This allows any desired pattern to be displayed.

When the beam reaches the end of a line, it is brought back to the beginning of the next line at a rate that is much faster than was used to generate the line. This action is referred to as "retrace". During the retrace period the electron beam is usually shut off so that it doesn't appear on the screen.

As the electron beam is moving across the screen horizontally, it is also moving downward. Because of this, each successive line starts slightly below the previous line. When the beam finally reaches the bottom right hand corner of the screen, it retraces vertically back to the top left hand corner. The time it takes for the beam to move from the top of the screen to the bottom and back again to the top is usually referred to as a "frame". In the United States, commercial television broadcast use 15,750 Hz as the horizontal sweep frequency (63.5 microseconds per horizontal line) and 60 Hz as the vertical sweep frequency or "frame" (16.67 milliseconds per vertical frame).

Although, the 60 Hz vertical frame and the 15,750 Hz horizontal line are the standards used by commercial broadcasts, they are by no means the only frequency at which CRTs can operate. In fact, many CRT displays use a horizontal scan that is around 18 KHz to 22 KHz and some even exceed 30 KHz. As the horizontal frequency increases, the number of horizontal lines per frame increases. Hence, the resolution on the vertical axis increases. This increased resolution is needed on high density graphic displays and on special text editting terminals that display many lines of text on the CRT.

Although many CRTs operate at non-standard horizontal frequencies, very few operate at vertical frequencies other than 60 Hz. If a vertical frequency other than 60 Hz is chosen, any external or internal magnetic or electrical variations at 60 Hz will modulate the electron beam and the image on the screen will be unstable. Since, in the United States, the power line frequency happens to be 60 Hz, there is a good chance for 60 Hz interference to exist. Transformers can cause 60 Hz magnetic fields and power supply ripple can cause 60 Hz electrical variations. To overcome this, special shielding and power supply regulation must be employed. In this design, we will assume a standard frame rate of 60 Hz and a standard line rate of 15,750 Hz.

By dividing the 63.5 microsecond horizontal line rate into the 16.67 millisecond vertical rate, it is found that there are 262.5 horizontal lines per vertical frame. At first, the half line may seem a bit odd, but actually it allows the resolution on the CRT to be effectively doubled. This is done by inserting a second set of horizontal lines between the first set (interlacing). In an interlaced system the line sets are not generated simultaneously. In a 60 Hz system, first all of the even-numbered lines are scanned: 0, 2, 4, . . . 524. Then all the odd-numbered lines: 1, 3, 5, . . . 525. Each set of lines usually contains different data (Figure 2.2).

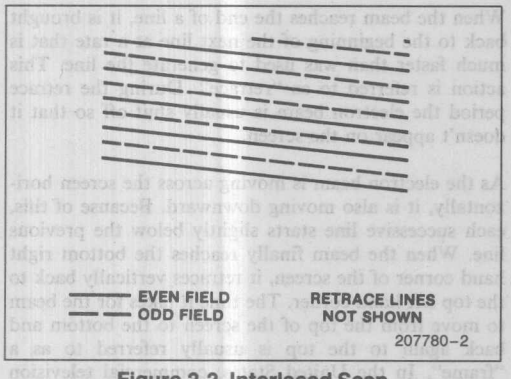


Figure 2-2. Interlaced Scan

Although interlacing provides greater resolution, it also has some distinct disadvantages. First of all, the circuitry needed to generate the extra half horizontal line per frame is quite complex when compared to a noninterlaced design, which requires an integer number of horizontal lines per frame. Next, the overall vertical refresh rate is half that of a noninterlaced display. As a result, flicker may result when the CRT uses high speed phosphors. To keep things as simple as possible, this design uses the noninterlaced approach.

The first thing any CRT controller must do is generate pulses that define the horizontal line timing and the vertical frame timing. This is usually done by dividing a crystal reference source by some appropriate numbers. On most raster scan CRTs the horizontal frequency is very forgiving and can vary by around 500 Hz or so and produce no ill effects. This means that the CRT itself can track a horizontal frequency between 15250 Hz and 16250 Hz, or in other words, there can be 256 to 270 horizontal lines per verticle frame. But, as mentioned earlier, the vertical freqency should be 60 Hz to insure stability.

The characters that are viewed on the screen are formed by a series of dots that are shifted out of the controller while the electron beam moves across the face of the CRT. The circuits that create this timing are referred to as the dot clock and character clock. The character clock is equal to the dot clock divided by the number of dots used to form a character along the horizontal axis and the dot clock is calculated by the following equation:

$$\text{DOT CLOCK (Hz)} = (N + R) * D * L * F$$

where N is the number of displayed characters per row,

R is the number of retrace character time increments,

D is the number of dots per character,

L is the number of horizontal lines per frame and

F is the frame rate in Hz.

In this design N = 80, R = 20, D = 7, L = 270, and F = 60 Hz. If the numbers are plugged in, the dot clock is found to be 11.34 MHz.

The retrace number, R, may vary from system to system because it is used to establish the margins on the left and right hand sides of the CRT. In this particular design R = 20 was empirically found to be optimum. The number of dots per character may vary depending on the character generator used and the number of dot clocks the designer wants to place between characters. This design uses a 5 × 7 dot matrix and allows 2 dot clock periods between characters (see Figure 2.3); since 5 + 2 equals 7, we find that D = 7.

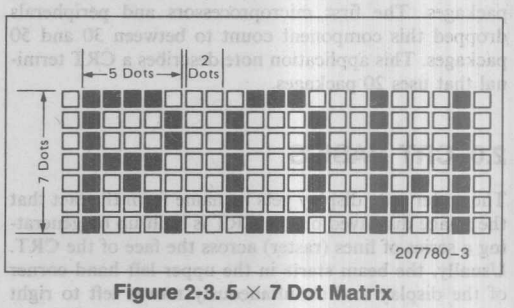


Figure 2-3. 5 × 7 Dot Matrix

the following equation:

$$L = (H * Z) + V$$

where, H is the number of horizontal lines per character,

Z is the number of character lines per frame and

V is the number of horizontal lines during vertical retrace. In this design, a 5 × 7 dot matrix is to be placed on a 7 × 10 field, so H = 10. Also, 25 lines are to be displayed, so Z = 25. As mentioned before, V = 20. When the numbers are plugged into the equation, L is found to be equal to 270 lines per frame.

The designer should be cautioned that these numbers are interrelated and that to guarantee proper operation 256 and 270. If L does not lie within these bounds the horizontal circuits of the CRT may not be able to lock onto the driving signal and the image will roll horizontally. The chosen L of 270 yields a horizontal frequency of 16,200 KHz on a 60 Hz frame and this number is within the 500 Hz tolerance mentioned earlier.

The V number is chosen to match the CRT in much the same manner as the R number mentioned earlier. When the electron beam reaches the bottom right corner of the screen it must retrace vertically to the top left corner. This retrace action requires time, usually between 900–1200 microseconds. To allow for this, enough horizontal sync times must be inserted during vertical retrace. Twenty horizontal sync times at

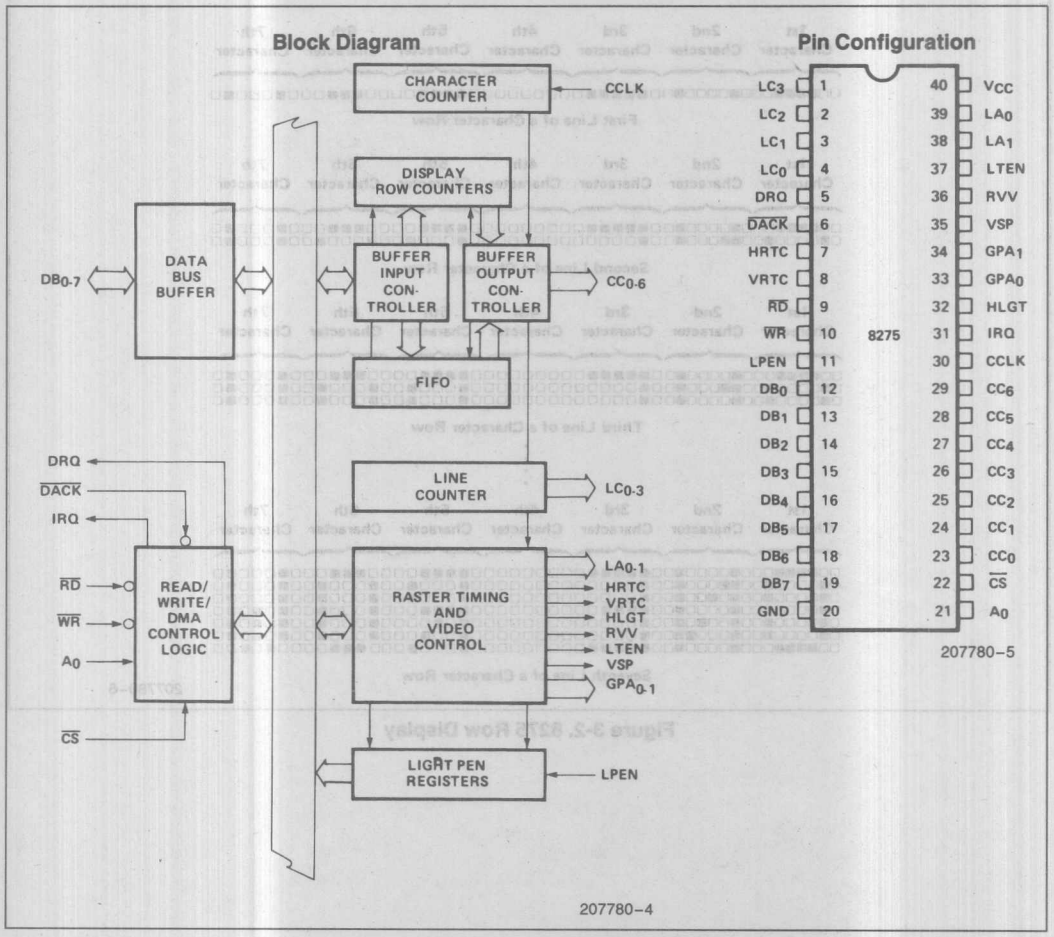


Figure 3-1. 8275 Block Diagram/Pin Configuration

61.5 microseconds yield a total of 1234.5 microseconds, which is enough time to allow the beam to return to the top of the screen.

The choices of H and Z largely relate to system design preference. As H increases, the character size along the vertical axis increases. Z is simply the number of lines of characters that are displayed and this, of course, is entirely a system design option.

3.0 8275 DESCRIPTION

A block diagram and pin configuration of the 8275 are shown in Figure 3.1. The following is a description of the general capabilities of the 8275.

3.1 CRT Display Refreshing

The 8275, having been programmed by the designer to a specific screen format, generates a series of DMA request signals, resulting in the transfer of a row of characters from display memory to the 8275's row buffers. The 8275 presents the character codes to an external character generator ROM by using outputs CC0-CC6. External dot timing logic is then used to transfer the parallel output data from the character generator ROM serially to the video input of the CRT. The character rows are displayed on the CRT one line at a time. Line count outputs LC0-LC3 are applied to the character generator ROM to perform the line selec-

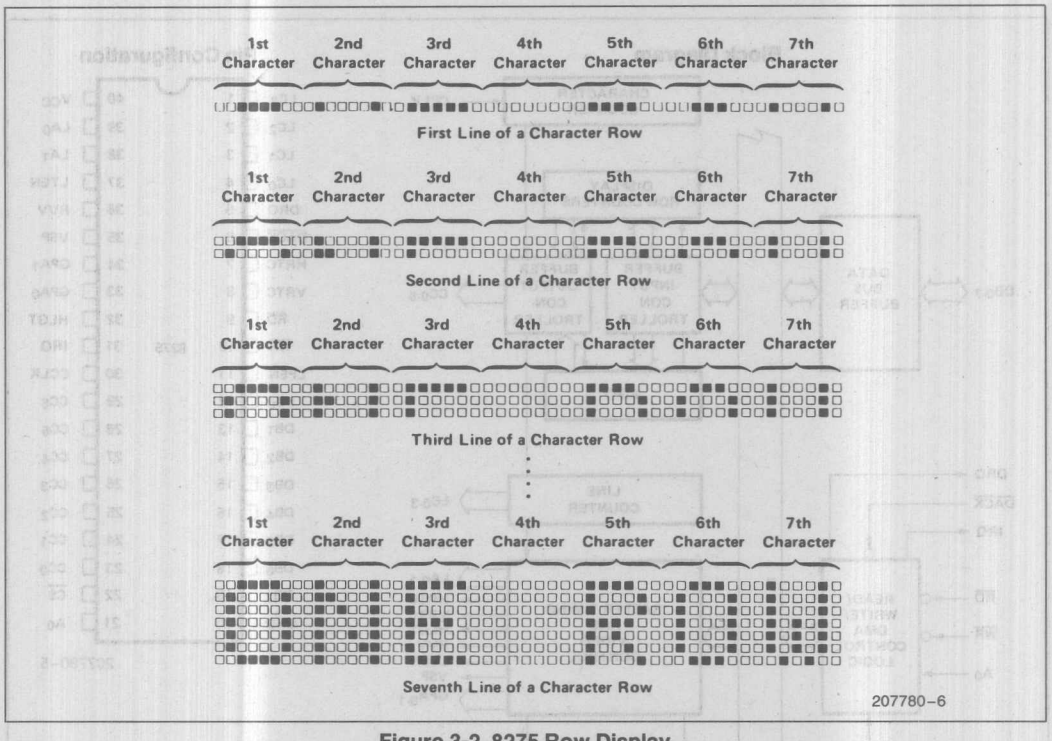


Figure 3-2. 8275 Row Display

tion function. The display process is illustrated in Figure 3.2. The entire process is repeated for each display row. At the beginning of the last displayed row, the 8275 issues an interrupt by setting the IRQ output line. The 8275 interrupt output will normally be connected to the interrupt input of the system central processor. The interrupt causes the CPU to execute an interrupt service subroutine. The service subroutine typically reinitializes DMA controller parameters for the next display refresh cycle, polls the system keyboard controller, and/or executes other appropriate functions. A block diagram of a CRT system implemented with the 8275 CRT Controller is provided in Figure 3.3. Proper CRT refreshing requires that certain 8275 parameters be programmed prior to the beginning of display operation. The 8275 has two types of programming registers, the Command Registers (CREG) and the Parameter Registers (PREG). It also has a Status Register (SREG). The Command Registers may only be written to and the Status Registers may only be read. The 8275 expects to recieve a command followed by a sequence of from 0 to 4 parameters, depending on the command. The 8275 instruction set consist of the eight commands shown in Figure 3.4.

To establish the format of the display, the 8275 provides a number of user programmable display format parameters. Display formats having from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per row are available.

In addition to transferring characters from memory to the CRT screen, the 8275 features cursor position control. The cursor position may be programmed, via X

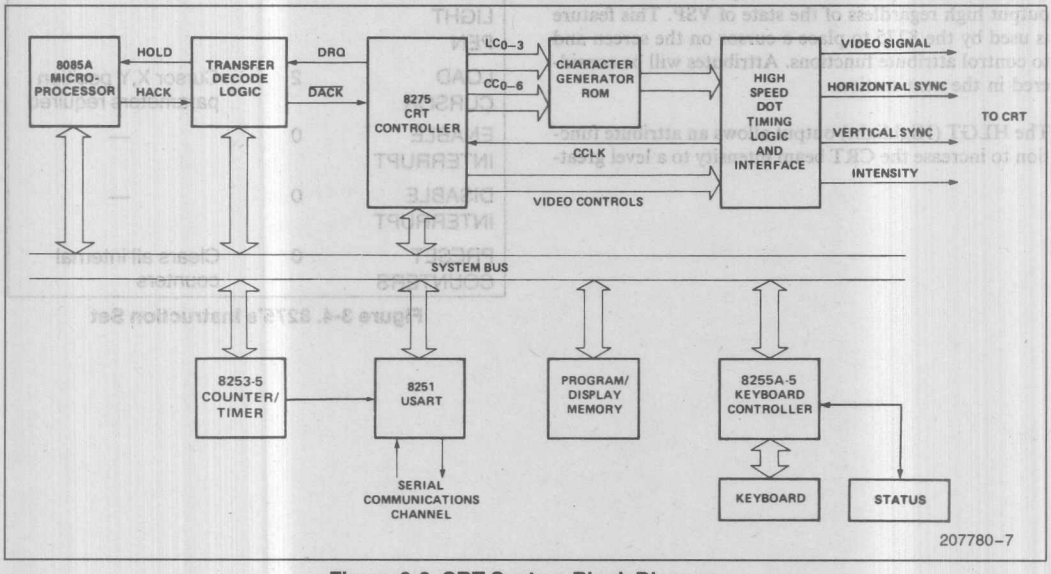


Figure 3-3. CRT System Block Diagram

and Y cursor position registers, to any character position on the display. The user may select from four cursor formats. Blinking or non-blinking underline and reverse video block cursors are available.

3.2 CRT Timing

The 8275 provides two timing outputs, HRTC and VRTC, which are utilized in synchronizing CRT horizontal and vertical oscillators to the 8275 refresh cycle. In addition, whenever HRTC or VRTC is active, a third timing output, VSP (Video Suppress) is true, providing a blinking signal to the dot timing logic. The dot timing logic will normally inhibit the video output to the CRT during the time when video suppress signal is true. An additional timing output, LTEN (Light Enable) is used to provide the ability to force the video output high regardless of the state of VSP. This feature is used by the 8275 to place a cursor on the screen and to control attribute functions. Attributes will be considered in the next section.

The HLGT (Highlight) output allows an attribute function to increase the CRT beam intensity to a level greater than normal. The fifth timing signal, RVV (Reverse Video) will, when enabled, cause the system video output to be inverted.

Command	No. of Parameter Bytes	Notes
RESET	4	Display format parameters required
START DISPLAY	0	DMA operation parameters included in command
STOP DISPLAY	0	—
READ LIGHT PEN	2	—
LOAD CURSOR	2	Cursor X,Y position parameters required
ENABLE INTERRUPT	0	—
DISABLE INTERRUPT	0	—
PRESET COUNTERS	0	Clears all internal counters

Figure 3-4. 8275's Instruction Set

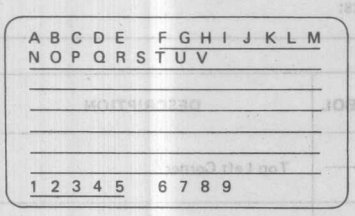
Character attributes were designed to produce the following graphics:

CHARACTER ATTRIBUTE CODE "CCCC"		OUTPUTS LA_1	OUTPUTS LA_0	OUTPUTS VSP	OUTPUTS LTEN	SYMBOL	DESCRIPTION
0000	Above Underline	0	0	1	0		Top Left Corner
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0001	Above Underline	0	0	1	0		Top Right Corner
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0010	Above Underline	0	1	0	0		Bottom Left Corner
	Underline	1	0	0	0		
	Below Underline	0	0	1	0		
0011	Above Underline	0	1	0	0		Bottom Right Corner
	Underline	1	1	0	0		
	Below Underline	0	0	1	0		
0100	Above Underline	0	0	1	0		Top Intersect
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
0101	Above Underline	0	1	0	0		Right Intersect
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0110	Above Underline	0	1	0	0		Left Intersect
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0111	Above Underline	0	1	0	0		Bottom Intersect
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1000	Above Underline	0	0	1	0		Horizontal Line
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1001	Above Underline	0	1	0	0		Vertical Line
	Underline	0	1	0	0		
	Below Underline	0	1	0	0		
1010	Above Underline	0	1	0	0		Crossed Lines
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
1011	Above Underline	0	0	0	0		Not Recommended *
	Underline	0	0	0	0		
	Below Underline	0	0	0	0		
1100	Above Underline	0	0	1	0		Special Codes
	Underline	0	0	1	0		
	Below Underline	0	0	1	0		
1101	Above Underline	Undefined					Illegal
	Underline						
	Below Underline						
1110	Above Underline	Undefined					Illegal
	Underline						
	Below Underline						
1111	Above Underline	Undefined					Illegal
	Underline						
	Below Underline						

207780-8

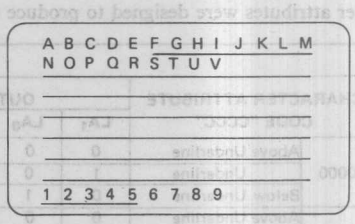
NOTES:
*Character Attribute Code 1011 is not recommended for normal operation. Sicne none of the attribute outputs are active, the character Generator will not be disabled, and an indeterminate character will be generated.
Character Attribute Codes 1101, 1110, and 1111 are illegal.
Blinking is active when B = 1.
Highlight is active when H = 1.

Figure 3-5. Character Attributes



207780-9

Example of the Visible Field Attribute Mode (Underline Attribute)



207780-10

Example of the Invisible Field Attribute Mode (Underline Attribute)

Figure 3-6. Field Attribute Examples

3.3 Special Functions

VISUAL ATTRIBUTES—Visual attributes are special codes which, when retrieved from display memory by the 8275, affect the visual characteristics of a character position or field of characters. Two types of visual attributes exist, character attributes and field attributes.

Character Attribute Codes: Character attribute codes can be used to generate graphics symbols without the use of a character generator. This is accomplished by selectively activating the Line Attribute outputs (LA0-LA1), the Video Suppression output (VSP), and the Light Enable output (LTEN). The dot timing logic uses these signals to generate the proper symbols. Character attributes can be programmed to blink or be highlighted individually. Blinking is accomplished with the Video Suppression output (VSP). Blink frequency is equal to the screen refresh frequency divided by 32. Highlighting is accomplished by activating the Highlight output (HGLT). Character attributes were designed to produce the graphic symbols shown in Figure 3.5.

Field Attribute Codes: The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the field attribute code up to, and including, the character which precedes the next field attribute code, or up to the end of the frame.

There are six field attributes:

1. *Blink*—Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2. *Highlight*—Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).
3. *Reverse Video*—Characters following the code are caused to appear in reverse video format by activating the Reverse Video output (RVV).
4. *Underline*—Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).
5. *General Purpose*—There are two additional 8275 outputs which act as general purpose, independently programmable field attributes. These attributes may be used to select colors or perform other desired control functons.

The 8275 can be programmed to provide visible or invisible field attribute characters as shown in Figure 3.6. If the 8275 is programmed in the visible field attribute mode, all field attributes will occupy a position on the screen. They will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character. If the 8275 is programmed in the invisible field attribute mode, the 8265 row buffer FIFOs are activated. The FIFOs effectively lengthen the row buffers by 16 characters, making room for up to 16 field attribute characters per display row. The FIFOs are 126 characters by 7 bits in size. When a field attribute is placed in the row buffer during DMA, the buffer input controller recognizes it and places the next character in the proper FIFO. When a field attribute is placed in the buffer output controller during display, it causes the controller to immediately put a character from the FIFO on the Character Code outputs (CC0-6). The chosen attributes are also activated.

LIGHT PEN DETECTION—A light pen consists fundamentally of a switch and light sensor. When the light pen is pressed against the CRT screen, the switch enables the light sensor. When the raster sweep coincides with the light sensor position on the display, the light pen output is input and the row and character position coordinates are stored in two 8275 internal registers. These registers can be read by the microprocessor.

SPECIAL CODES—Four special codes may be used to help reduce memory, software, or DMA overhead. These codes are placed in character positions in display memory.

1. *End of Row Code*—Activates VSP. VSP remains active until the end of the line is reached. While VSP is active, the screen is blanked.
2. *End of Row-Stop DMA Code*—Causes the DMA Control Logic to stop DMA for the rest of the row when it is written into the row buffer. It affects the display in the same way as the End of Row Code.
3. *End of Screen Code*—Activates VSP. VSP remains active until the end of the frame is reached.
4. *End of Screen-Stop DMA Code*—Causes the DMA Control Logic to stop DMA for the rest of the frame when it is written into the row buffer. It affects the display in the same way as the End of Screen Code.

PROGRAMMABLE DMA BURST CONTROL—The 8275 can be programmed to request single-byte DMA transfers of DMA burst transfers of 2, 4, or 8 characters per burst. The interval between bursts is also programmable. This allows the user to tailor the DMA overhead to fit the system needs.

4.0 DESIGN BACKGROUND

4.1 Design Philosophy

Since the cost of any CRT system is somewhat proportional to parts count, arriving at a minimum part count solution without sacrificing performance has been the motivating force throughout this design effort. To successfully design a CRT terminal and keep the parts count to a minimum, a few things became immediately apparent.

1. An 8085 should be used.
2. Address and data buffering should be eliminated.
3. Multi-port memory should be eliminated.
4. DMA should be eliminated.

Decision 1 is obvious, the 8085's on-board clock generator, bus controller and vectored interrupts greatly reduce the overall part count considerably. Decision 2 is fairly obvious; if a circuit can be designed so that loading on the data and address lines is kept to a minimum, both the data and address buffers can be eliminated. This easily saves three to eight packages and reduces the power consumption of the design. Both decisions 3 and 4 require a basic understanding of current CRT design concepts.

In any CRT design, extreme time conflicts are created because all essential elements require access to the bus. The CPU needs to access the memory to control the system and to handle the incoming characters, but, at the same time, the CRT controller needs to access the memory to keep the raster scan display refreshed. To resolve this conflict two common techniques are employed, page buffering and line buffering.

In the page buffering approach the entire screen memory is isolated from the rest of the system. This isolation is usually accomplished with three-state buffers or two line to one line multiplexers. Of course, whenever a character needs to be manipulated the CPU must gain access to the buffered memory and, again, possible contention between the CPU and the CRT controller results. This contention is usually resolved in one of two ways: (1) the CPU is always given priority, or; (2) the CPU is allowed to access the buffered memory only during horizontal and vertical retrace times.

Approach 1 is the easiest to implement from a hardware point of view, but if the CPU always has priority the display may temporarily blink or "flicker" while the CPU accesses the display memory. This, of course, occurs because when the CPU accesses the display memory the CRT controller is not able to retrieve a character, so the display must be blanked during this time. Aesthetically, this "flickering" is not desirable, so approach 2 is often used.

The second approach eliminates the display flickering encountered in the previously mentioned technique, but additonal hardware is required. Usually the vertical and horizontal blank signals are gated with the buffered memory select lines and this line is used to control the CPU's ready line. So, if the CPU wants to use the buffered memory, its ready line is asserted until horizontal or vertical retrace times. This, of course, will impact the CPU's overall through put.

Both page buffered approaches require a significant amount of additional hardware and for the most part are not well suited for a minimum parts count type of terminal. This guides us to the line buffered approach. This approach eliminates the separate buffered memory for the display, but, at the same time, introduces a few new problems that must be solved.

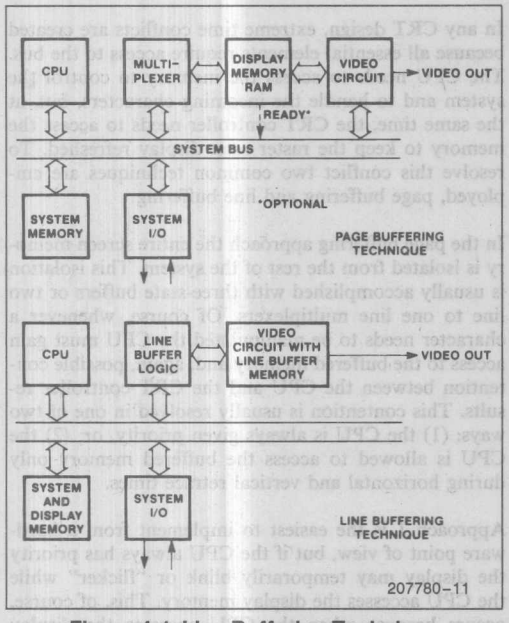


Figure 4-1. Line Buffering Technique

```
CLOCK CYCLES    SEQ     SOURCE STATEMENT

    10          1       PUSH    PSW         ;SAVE A AND FLAGS
    10          2       PUSH    H           ;SAVE H AND L
    10          3       PUSH    D           ;SAVE D AND E
    10          4       LXI     H,PORCH     ;ZERO H AND L
    10          5       DAD     SP          ;PUT STACK POINTER IN H AND L
     4          6       XCHG                ;PUT STACK IN D AND E
    16          7       LHLD    CURAD       ;GET POINTER
     6          8       SPHL                ;PUT CURSOR LINE INTO SP
     7          9       MVI     A,80H       ;SET MASK FOR SIM
     4          10      SIM                 ;CLR SPECIAL TRANSFER BIT
   400          11      POP     H           ;DO 40 POPS
     4          12      RFC                 ;SET UP A
     4          13      SIM                 ;GO BACK TO NORMAL MODE
    10          14      LXI     H,0000H     ;ZERO HL
    10          15      DAD     SP          ;ADD STACK
     4          16      XCHG                ;PUT STACK IN H AND L
     6          17      SPHL                ;RESTORE STACK
    10          18      LXI     H,LAST      ;PUT BOTTOM DISPLAY IN H AND L
     4          19      XCHG                ;SWAP REGISTERS
     4          20      MOV     A,D         ;PUT HIGH ORDER IN A
     4          21      CMP     H           ;SEE IF SAME AS H
   7/10         22      JNZ     XPTR        ;IF NOT LEAVE
     4          23      MOV     A,E         ;PUT LOW ORDER IN A
     4          24      CMP     L           ;SEE IF SAME AS L
   7/10         25      JNZ     XPTR        ;IF NOT LEAVE
    10          26      LXI     H,TPOS      ;LOAD H AND L WITH TOP OF SCREEN MEMORY
    16          27 XPTR: SHLD   CURAD       ;PUT BACK CURRENT ADDRESS
     7          28      MVI     A,10H       ;GET MASK BYTE
     4          29      SIM                 ;SET INTERRUPT MASK
    10          30      POP     D           ;GET D AND E
    10          31      POP     H           ;GET H AND L
    10          32      POP     PSW         ;GET A AND FLAGS
     4          33      EI                  ;ENABLE INTERRUPTS
    10          34      RET                 ;GO BACK

    TOTAL CLOCK CYCLES = 650 (WORST CASE)

    WITH A 6.144 MHZ CRYSTAL TOTAL TIME TO FILL

    ROW BUFFER ON 8275 = 650 * .325 = 211.25 MICROSECONDS
```

207780-12

Figure 4-2. Routine To Load 8275's Row Buffers

In the line buffered approach both the CPU and the CRT controller share the same memory. Every time the CRT controller needs a new character or line of data, normal processing activity is halted and the CRT controller accesses memory and displays the data. Just how the CRT controller needs to acquire the display data greatly affects the performance of the overall system. Whether the CRT controller needs to gain access to the main memory to acquire a single character or a complete line of data depends on the presence or absence of a separate line or row buffer.

If no row buffer is present the CRT controller must go to the main memory to fetch every character. This of course, is not a very efficient approach because the processor will be forced to relinquish the bus 70% to 80% of the time. So much processor inactivity greatly affects the overall system performance. In fact terminals that use this approach are typically limited to around 1200 to 2400 baud on their serial communication channels. This low baud rate is in general not acceptable, hence this approach was not chosen.

If a separate row buffer is employed the CRT controller only has to access the memory once for each displayed character per line. This forces the processor to relinquish the bus only about 20% to 35% of the time and a full 4800 to 9600 baud can be achieved. Figure 4.1 illustrates these different techniques.

The 8275 CRT controller is ideal for implementing the row buffer approach because the row buffer is contained on the device itself. In fact, the 8275 contains two 80-byte row buffers. The presence of two row buffers allow one buffer to be filled while the other buffer is displaying the data. This dual row buffer approach enhances CPU performance even further.

4.2 Using the 8275 without DMA

Until now the process of filling the row buffer has only been alluded to. In reality, a DMA technique is usually used. This approach was demonstrated in AP-32 where an 8257 DMA controller was mated to an 8275 CRT controller. In order to minimize component count, this design eliminates the DMA controller and its associated circuitry while replacing them with a special interrupt-driven transfer.

The only real concern with using the 8275 in an interrupt-driven transfer mode is speed. Eighty characters must be loaded into the 8275 every 617 microseconds and the processor must also have time to perform all the other tasks that are required. To minimize the overhead associated with loading the characters into the 8275 a special technique was employed. This technique involves setting a special transfer bit and executing a string of POP instructions. The string of POP instruc-

tions is used to rapidly move the data from the memory into the 8275. Figure 4.2 shows the basic software structure.

In this design the 8085's SOD line was used as the special transfer bit. In order to perform the transfer properly this special bit must do two things: (1) turn processor reads into $\overline{DACK}$ plus $\overline{WR}$ for the 8275; and (2) mask processor fetch cycles from the 8275, so that a fetch cycle does not write into the 8275. Conventional logic could have been used to implement this special function, but in this design a small bipolar programmable read only memory was used. Figure 4.3 shows a basic version of the hardware.

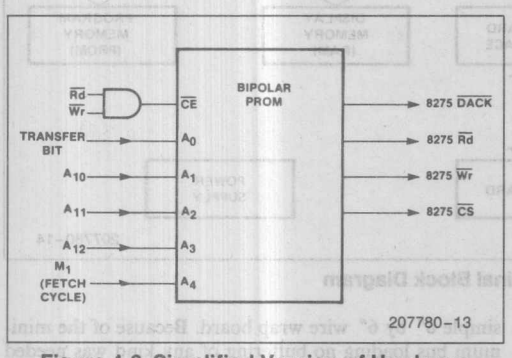


Figure 4-3. Simplified Version of Hardware Decoder

At first, it may seem strange that we are supplying a $\overline{DACK}$ when no DMA controller exists in the system. But the reader should be aware that all Intel peripheral devices that have DMA lines actually use $\overline{DACK}$ as a chip select for the data. So, when you want to write a command or read status you assert $\overline{CS}$ and $\overline{WR}$ or $\overline{RD}$, but when you want to read or write data you assert $\overline{DACK}$ and $\overline{RD}$ or $\overline{WR}$. The peripheral device doesn't "know" if a DMA controller is in the circuit or not. In passing, it should be mentioned that $\overline{DACK}$ and $\overline{CS}$ should not be asserted on the same device at the same time, since this combination yields an undefined result.

This POP technique actually compares quite favorably in terms of time to the DMA technique. One POP instruction transfers two bytes of data to the 8275 and takes 10 CPU clock cycles to execute, for a net transfer rate of one byte every five clock cycles. The DMA controller takes four clock cycles to transfer one byte but, some time is lost in synchronization. So the difference between the two techniques is one clock cycle per byte maximum. If we compare the overall speed of the 8085 to the speed of the 8080 used in AP-32, we find that at 3 MHz we can transfer one byte every 1.67 microseconds using the 8085 and POP technique vs. 2 microseconds per byte for the 2 MHz 8080 using DMA.

5.0 CIRCUIT DESCRIPTION

5.1 Scope of the Project

A fully functional, microprocessor-based CRT terminal was designed and constructed using the 8275 CRT controller and the 8085 as the controlling element. The terminal had many of the functions found in existing commercial low-cost terminals and more sophisticated features could easily be added with a modest amount of additional software. In order to minimize component count LSI devices were used whenever possible and software was used to replace hardware.

5.2 System Target Specifications

The design specifications for the CRT terminal were as follows:

Display Format

- 80 characters per display row
- 25 display rows

Character Format

- 5 × 7 dot matrix character contained within a 7 × 10 matrix
- First and seventh columns blanked
- Ninth line cursor position
- Blinking underline cursor

Special Characters Recognized

- Control characters
- Line feed
- Carriage Return
- Backspace
- Form feed

Escape Sequences Recognized

- ESC, A, Cursor up
- ESC, B, Cursor down
- ESC, C, Cursor right
- ESC, D, Cursor left
- ESC, E, Clear screen
- ESC, H, Home cursor
- ESC, J, Erase to the end of the screen
- ESC, K, Erase to the current line

Characters Displayed

- 96 ASCII alphanumeric characters
- Special control characters

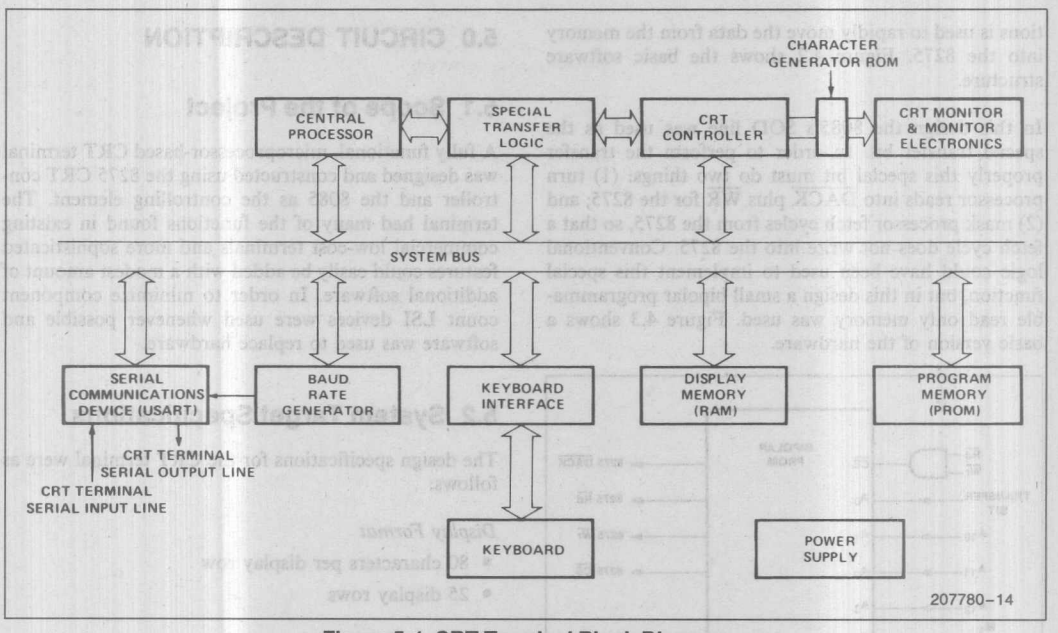


Figure 5-1. CRT Terminal Block Diagram

Characters Transmitted

- 96 ASCII alphanumeric characters
- ASCII control characters

Program Memory

- 2K bytes of 2716 EPROM

Display/Buffer/Stack Memory

- 2K bytes 2114 static memory (4 packages)

Data Rate

- 9600 BAUD using 3 MHz 8085

CRT Monitor

- Ball Bros TV-12, 12 MHz B.W.

Keyboard

- Any standard un-encoded ASCII keyboard

Screen Refresh Rate

- 60 Hz

5.3 Hardware Description

A block diagram of the CRT terminal is shown in Figure 5.1. The diagram shows only the essential system features. A detailed schematic of the CRT is contained in the Appendix. The terminal was constructed on a simple 6″ by 6″ wire wrap board. Because of the minimum bus loading no buffering of any kind was needed (see Figure 5.2).

Worst case bus loading:

Data Bus:	8275	20 pF
	8255A-5	20 pF
	8253-5	20 pF
	8253-5	20 pF
	8251A	20 pF
	2×2114	10 pF
	2716	12 pF
	8212	12 pF
		114 pF max

Only A_8–A_{15} are important since A_0–A_7 are latched by the 8212

Address Bus:	4×2114	20 pF
	2716	6 pF
		26 pF max

This loading assures that all components will be compatible with 3 MHz 8085 and that no wait states will be required

Figure 5-2. Bus Loading

The "heart" of the CRT terminal is the 8085 microprocessor. The 8085 initializes all devices in the system, loads the CRT controller, scans the keyboard, assembles the characters to be transmitted, decodes the in-

coming characters and determines where the character is to be placed on the screen. Clearly, the processor is quite busy.

A standard list of LSI peripheral devices surround the 8085. The 8251A is used as the serial communication link, the 8255A-5 is used to scan the keyboard and read the system variables through a set of switches, and the 8253 is used as a baud rate generator and as a "horizontal pulse extender" for the 8275.

The 8275 is used as the CRT controller in the system, and a 2716 is used as the character generator. To handle the high speed portion of the terminal the 8275 is surrounded by a small handfull of TTL. The program memory is contained in one 2716 EPROM and the data and screen memory use four 2114-type RAMs.

All devices in this system are memory mapped. A bipolar PROM is used to decode all of the addresses for the RAM, ROM, 8275, and the 8253. As mentioned earlier, the bipolar prom also turns $\overline{\text{READ}}$s into $\overline{\text{DACK}}$'s and $\overline{\text{WR}}$'s for the 8275. The 8255 and 8253 are decoded by a simple address line chip select method. The total package count for the system is 20, not including the serial line drivers. If this same terminal were designed using the MCS-85 family of integrated circuits, additional part savings could have been realized. The four 2114's could have been replaced by two 8185's and the 8255 and the 2716 program PROM could have been replaced by one 8755. Additionally, since both the 8185 and the 2716 have address latches no 8212 would be needed, so the total parts count could be reduced by three or four packages.

5.4 System Operation

The 8085 CPU initializes each peripheral to the appropriate mode of operation following system reset. After initialization, the 8085 continually polls the 8251A to see if a character has been sent to the terminal. When a character has been received, the 8085 decodes the character and takes appropriate action. While the 8085 is executing the above "foreground" programs, it is being interrupted once every 617 microseconds by the 8275. This "background" program is used to load the row buffers on the 8275. The 8085 is also interrupted once every frame time, or 16.67 ms, to read the keyboard and the status of the 8275.

As discussed earlier, a special POP technique was used to rapidly move the contents of the display RAM into the 8275's row buffers. The characters are then synchronously transferred to the character code outputs CC0–CC6, connected to the character generator address lines A3–A9 (Figure 5.3). Line count outputs LC0–LC2 from the 8275 are applied to the character generator address lines A0–A2. The 8275 displays character rows one line at a time. The line count outputs are used to determine which line of the character selected by A3–A8 will be displayed. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line of the character row is selected. This process continues until the last line of the row is transferred to the dot timing logic.

The dot timing logic latches the output of the character generator ROM into a parallel in, serial out sysnchronous shift register. This shift register is clocked at the dot clock rate (11.34 MHz) and its output constitutes the video input to the CRT.

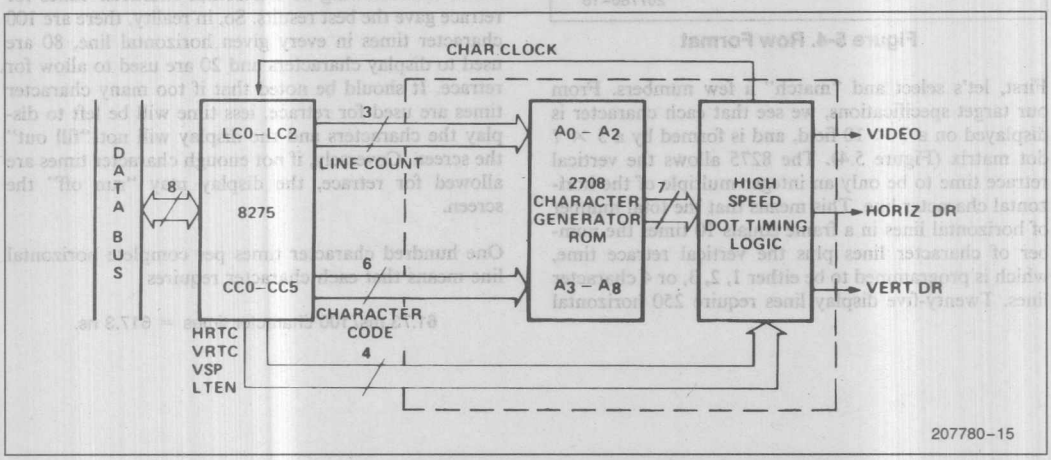


Figure 5-3. Character Generator/Dot Timing Logic Block Diagram

Table 5-1

Parameter	Range
Vertical Blanking Time (VRTC)	900 μs nominal
Vertical Drive Pulsewidth	300 μs ≤ PW ≤ 1.4 ms
Horizontal Blanking Time (HRTC)	11 μs nominal
Horizontal Drive Pulsewidth	25 μs ≤ PW ≤ 30 μs
Horizontal Repetition Rate	15,750 ±500 pps

5.5 System Timing

Before any specific timing can be calculated it is necessary to determine what constraints the chosen CRT places on the overall timing. The requirements for the Ball Bros. TV-12 monitor are shown in Table 5.1. The data from Table 5.1, the 8275 specifications, and the system target specifications are all that is needed to calculate the system's timing.

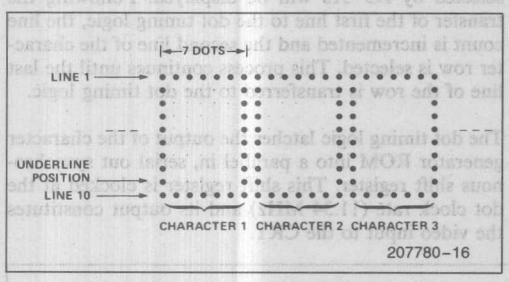


Figure 5-4. Row Format

First, let's select and "match" a few numbers. From our target specifications, we see that each character is displayed on a 7 × 10 field, and is formed by a 5 × 7 dot matrix (Figure 5.4). The 8275 allows the vertical retrace time to be only an integer multiple of the horizontal character line. This means that the total number of horizontal lines in a frame equals 10 times the number of character lines plus the vertical retrace time, which is programmed to be either 1, 2, 3, or 4 character lines. Twenty-five display lines require 250 horizontal lines. So, if we wish to have a horizontal frequency in the neighborhood of 15,750 Hz we must choose either one or two character lines for vertical retrace. To allow for a little more margin at the top and bottom of the screen, two character lines were chosen for vertical retrace. This choice yields a net 250 + 20 = 270 horizontal lines per frame. So, assuming a 60 Hz frame:

60 Hz * 270 = 16,200 Hz (horizontal frequency)

This value falls within our target specification of 15,750 Hz with a 500 Hz variation and also assures timing compatibility with the Ball monitor since, 20 horizontal sync times yield a vertical retract time of:

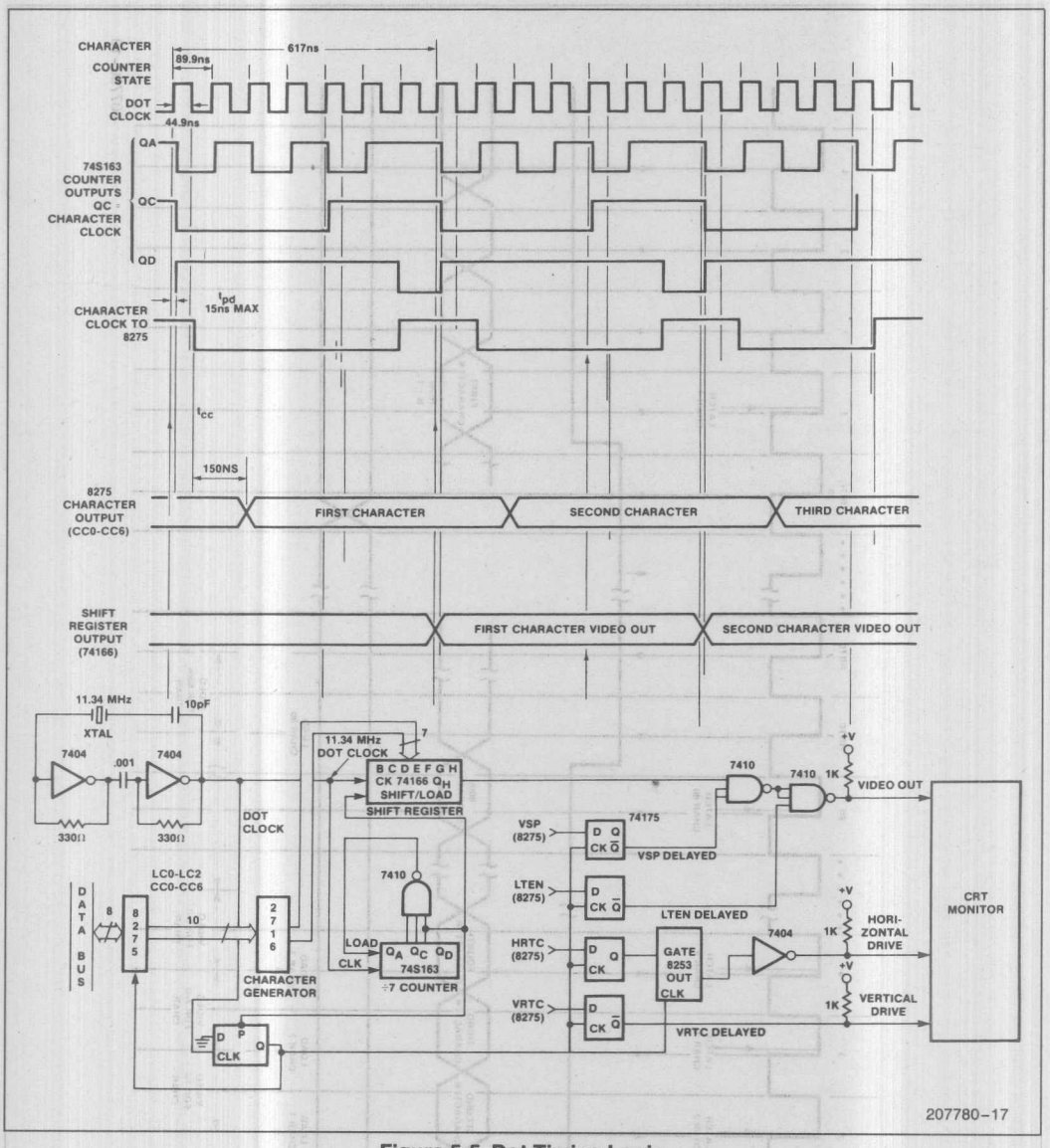
61.7 ms X 20 horizontal sync times = 1.2345 ms

This number meets the nominal VRTC and vertical drive pulse width time for the Ball monitor. A horizontal frequency of 16,200 Hz implies a 1/16,200 = 61.73 microsecond period.

It is now known that the terminal is using 250 horizontal lines to display data and 20 horizontal lines to allow for vertical retrace and that the horizontal frequency is 16,200 Hz. The next thing that needs to be determined is how much time must be allowed for horizontal retrace. Unfortunately, this number depends almost entirely on the monitor used. Usually, this number lies somewhere between 15 and 30 percent of the total horizontal line time, which in this case is 1/16,200 Hz or 61.73 microseconds. Since in most designs a fixed number of characters can be displayed on a horizontal line, it is often useful to express retrace as a given number of character times. In this design, 80 characters can be displayed on a horizontal line and it was empirically found that allowing 20 horizontal character times for retrace gave the best results. So, in reality, there are 100 character times in every given horizontal line, 80 are used to display characters and 20 are used to allow for retrace. It should be noted that if too many character times are used for retrace, less time will be left to display the characters and the display will not "fill out" the screen. Coversely, if not enough character times are allowed for retrace, the display may "run off" the screen.

One hundred character times per complete horizontal line means that each character requires

61.73 ms/100 character times = 617.3 ns.



207780–17

Figure 5-5. Dot Timing Logic

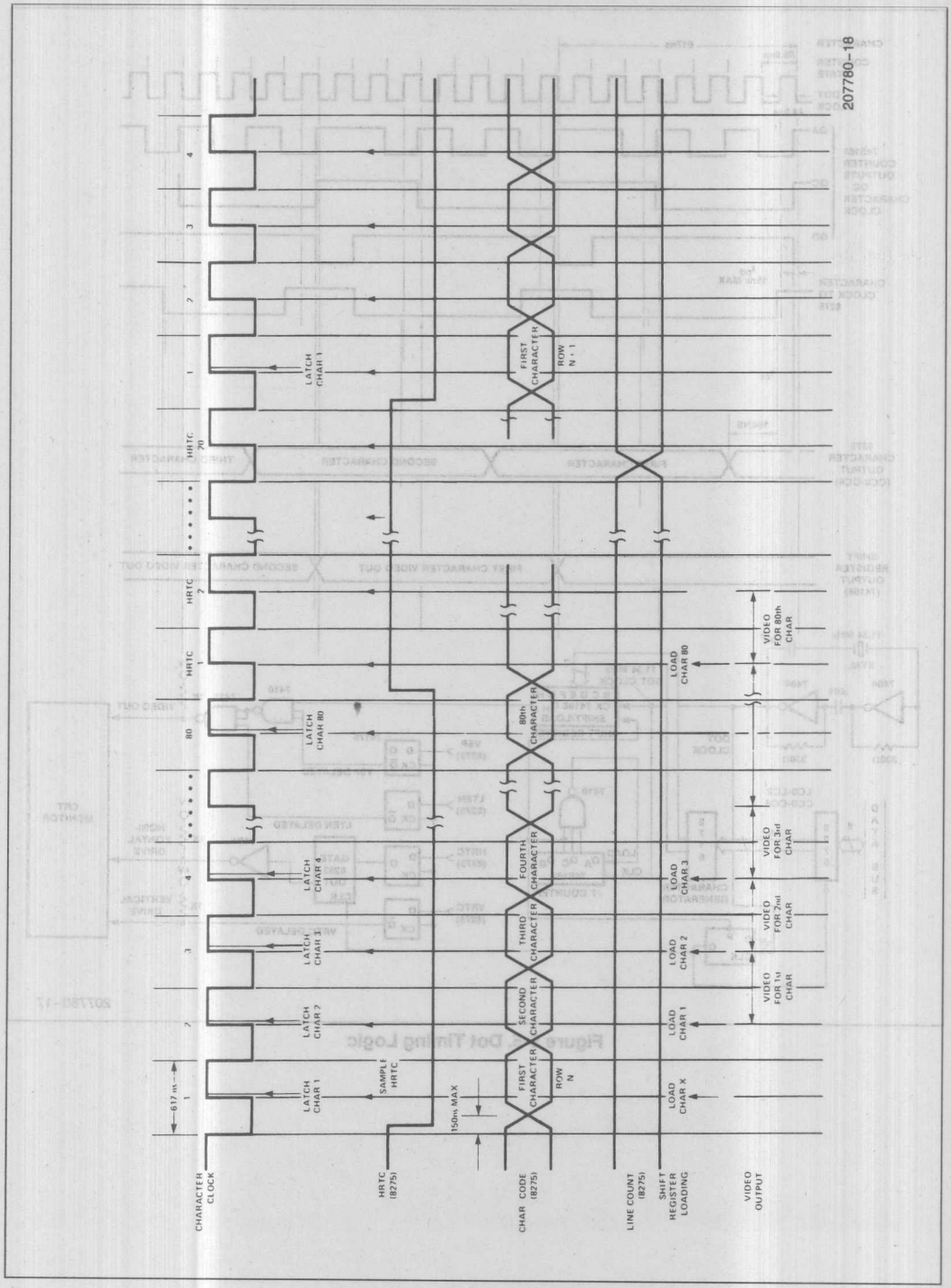


Figure 5-6. CRT System Timing

If we multiply the 20 horizontal retrace times by the 617.3 nanoseconds needed for each character, we find

617.3 ns * 20 retrace times = 12.345 ms

This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, an 8253 was programmed in the one-shot mode and was used to extend the horizontal drive pulsewidth.

Now that the 617.3 nanosecond character clock period is known, the dot clock is easy to calculate. Since each character is formed by placing 7 dots along the horizontal.

DOT CLOCK PERIOD = 617.3 ns
(CHARACTER CLK PERIOD)/ 7 DOTS

DOT CLOCK PERIOD = 88.183 nanoseconds

DOT CLOCK FREQUENCY = 1/PERIOD = 11.34 MHz

Figures 5.5 and 5.6 illustrate the basic dot timing and the CRT system timing, respectively.

6.0 SYSTEM SOFTWARE

6.1 Software Overview

As mentioned earlier the software is structured on a "foreground-background" basis. Two interrupt-driven routines, FRAME and POPDAT (Figure 6.1) request service every 16.67 milliseconds and 617 microseconds respectively, frame is used to check the baud rate switches, update the system pointers and decode and assemble the keyboard characters. POPDAT is used to move data from the memory into the 8275's row buffer rapidly.

The foreground routine first examines the line-local switch to see whether to accept data from the USART or the keyboard. If the terminal is in the local mode, action will be taken on any data that is entered through the keyboard and the USART will be ignored on both output and input. If the terminal is in the line mode data entered through the keyboard will be transmitted by the USART and action will be taken on any data read out of the USART.

When data has been entered in the terminal the software first determines if the character received was an escape, line feed, form feed, carriage return, back space, or simply a printable character. If an escape was received the terminal assumes the next received character will be a recognizable escape sequence character. If it isn't no operation is performed.

After the character is decoded, the processor jumps to the routine to perform the required task. Figure 6.2 is a flow chart of the basic software operations; the program is listed in Adppendix 6.8.

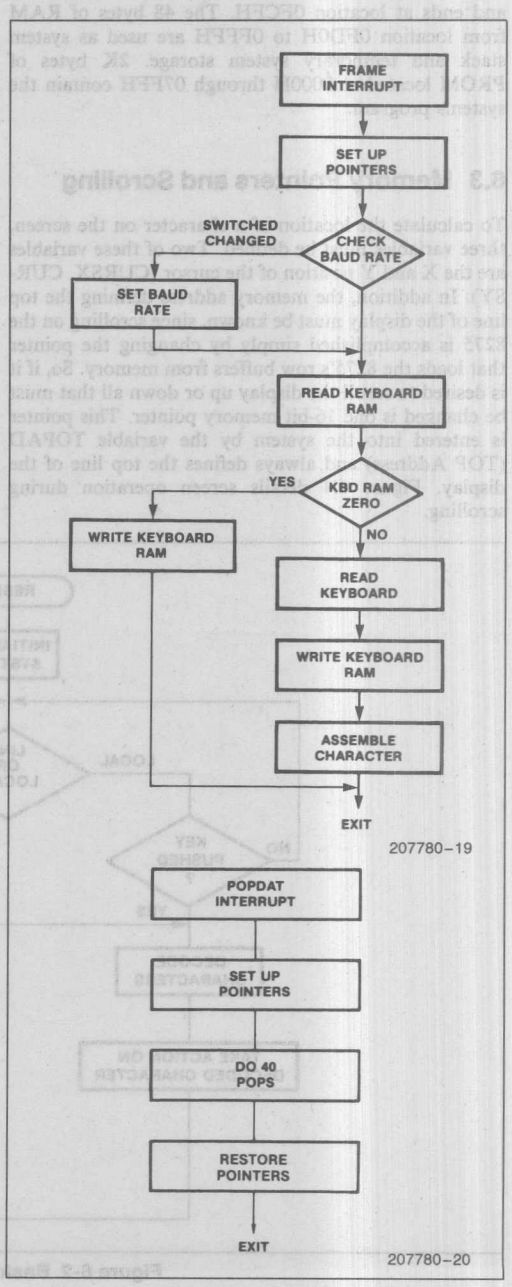


Figure 6-1. Frame and POPDAT Interrupt Routines

6.2 System Memory Organization

The display memory organization is shown in Figure 6.3. The display begins at location 0800H in memory and ends at location 0FCFH. The 48 bytes of RAM from location 0FD0H to 0FFFH are used as system stack and temporary system storage. 2K bytes of PROM located at 0000H through 07FFH contain the systems program.

6.3 Memory Pointers and Scrolling

To calculate the location of a character on the screen, three variables must be defined. Two of these variables are the X and Y position of the cursor (CURSX, CURSY). In addition, the memory address defining the top line of the display must be known, since scrolling on the 8275 is accomplished simply by changing the pointer that loads the 8275's row buffers from memory. So, if it is desired to scroll the display up or down all that must be changed is one 16-bit memory pointer. This pointer is entered into the system by the variable TOPAD (TOP Address) and always defines the top line of the display. Figure 6.4 details screen operation during scrolling.

	1st Column	2nd Column ..... 80th Column
ROW 1	0800H	0801H 084FH
ROW 2	0850H	0851H 089FH
ROW 3	08A0H	08A1H 08EFH
ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH

Figure 6-3. Screen Display After Initialization

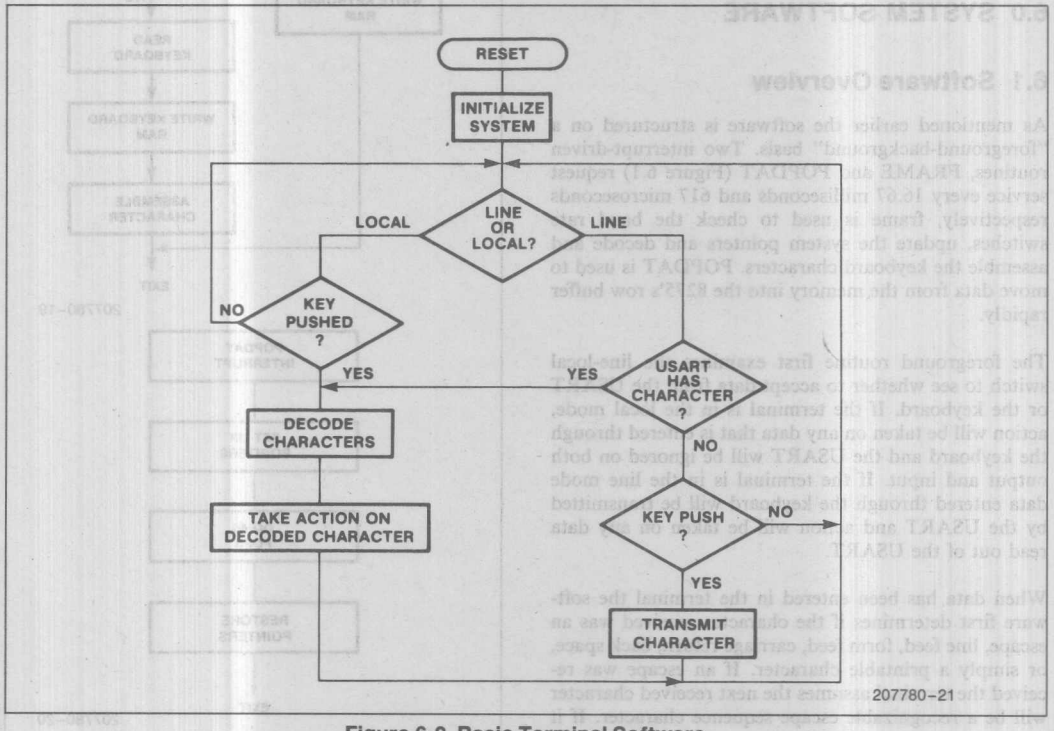


Figure 6-2. Basic Terminal Software

Subroutines CALCU (Calculate) and ADX (ADd X axis) use these three variables to calculate an absolute memory address. The subroutine CALCU is used whenever a location in the screen memory must be altered.

6.4 Software Timing

One important question that must be asked about the terminal software is, "How fast does it run". This is important because if the terminal is running at 9600 baud, it must be able to handle each received character in 1.04 milliseconds. Figure 6.5 is a flowchart of the subroutine execution times. It should be pointed out that all of the times listed are "worst case" execution times. This means that all routines assume they must do the maximum amount of data manipulation. For instance, the PUT routine assumes that the character is being placed in the last column and that a line feed must follow the placing of the character on the screen.

How fast do the routines need to execute in order to assure operation at 9600 baud? Since POPDAT interrupts occur every 617 microseconds, it is possible to receive two complete interrupt requests in every character time (1042 microseconds) at 9600 baud. Each POPDAT interrupt executes in 211 microseconds maximum. This means that each routine must execute in:

$$1042 - 2 * 211 = 620 \text{ microseconds}$$

By adding up the times for any loop, it is clear that all routines meet this speed requirement, with the exception of ESC J. This means that if the terminal is operating at 9600 baud, at least one character time must be inserted after an ESC J sequence.

ROW 1	0800H	0801H 084FH
ROW 2	0850H	0851H 089FH
ROW 3	08A0H	08A1H 08EFH
ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH

After Initialization

ROW 2	0850H	0851H 089FH
ROW 3	08A0H	08A1H 08EFH
ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH
ROW 1	0800H	0801H 084FH

After 1 Scroll

ROW 3	08A0H	08A1H 08EFH
ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH
ROW 1	0800H	0801H 084FH
ROW 2	0850H	0851H 089FH

After 2 Scrolls

ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH
ROW 1	0800H	0801H 084FH
ROW 2	0850H	0851H 089FH
ROW 3	08A0H	08A1H 08EFH

After 3 Scrolls

Figure 6-4. Screen Memory During Scrolling

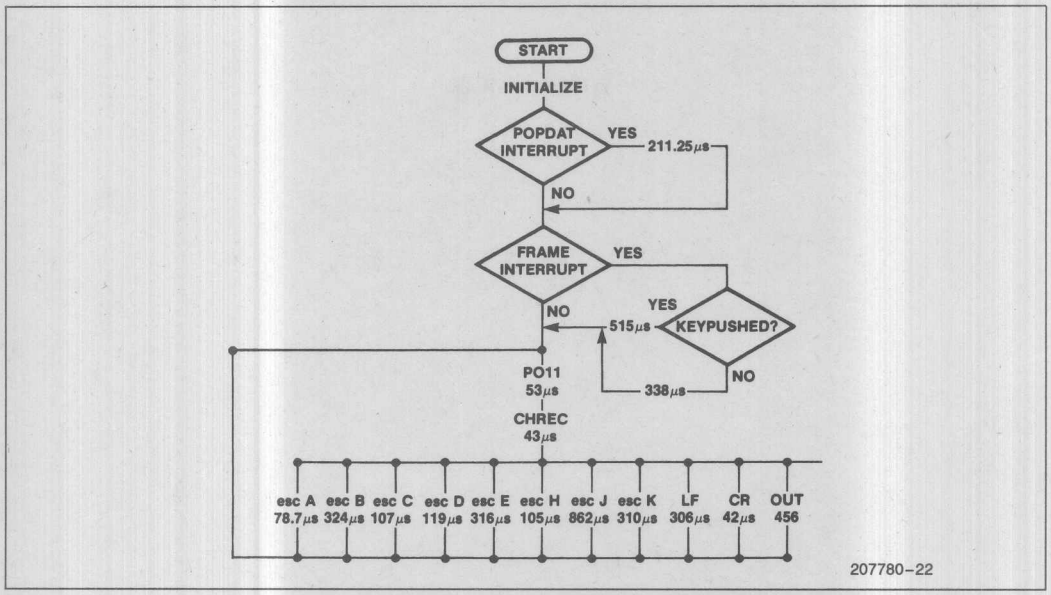
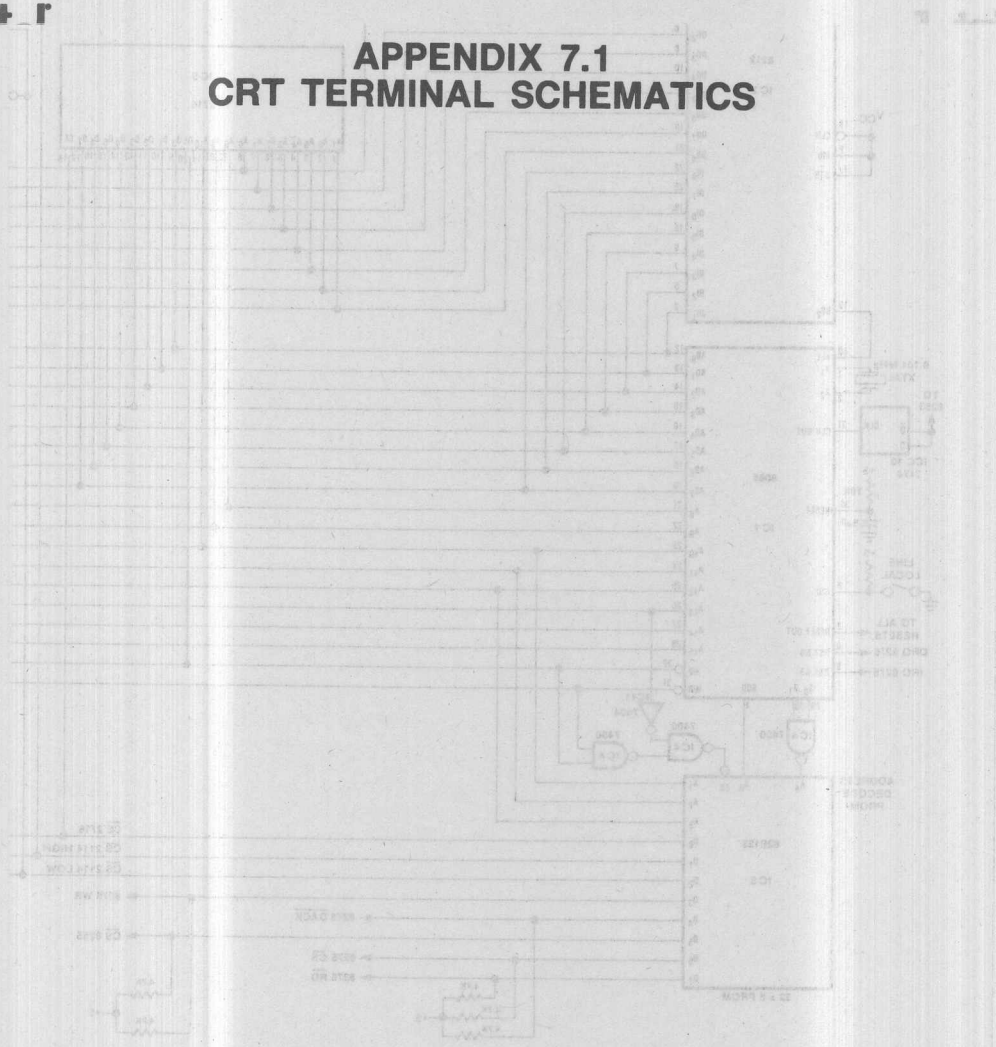
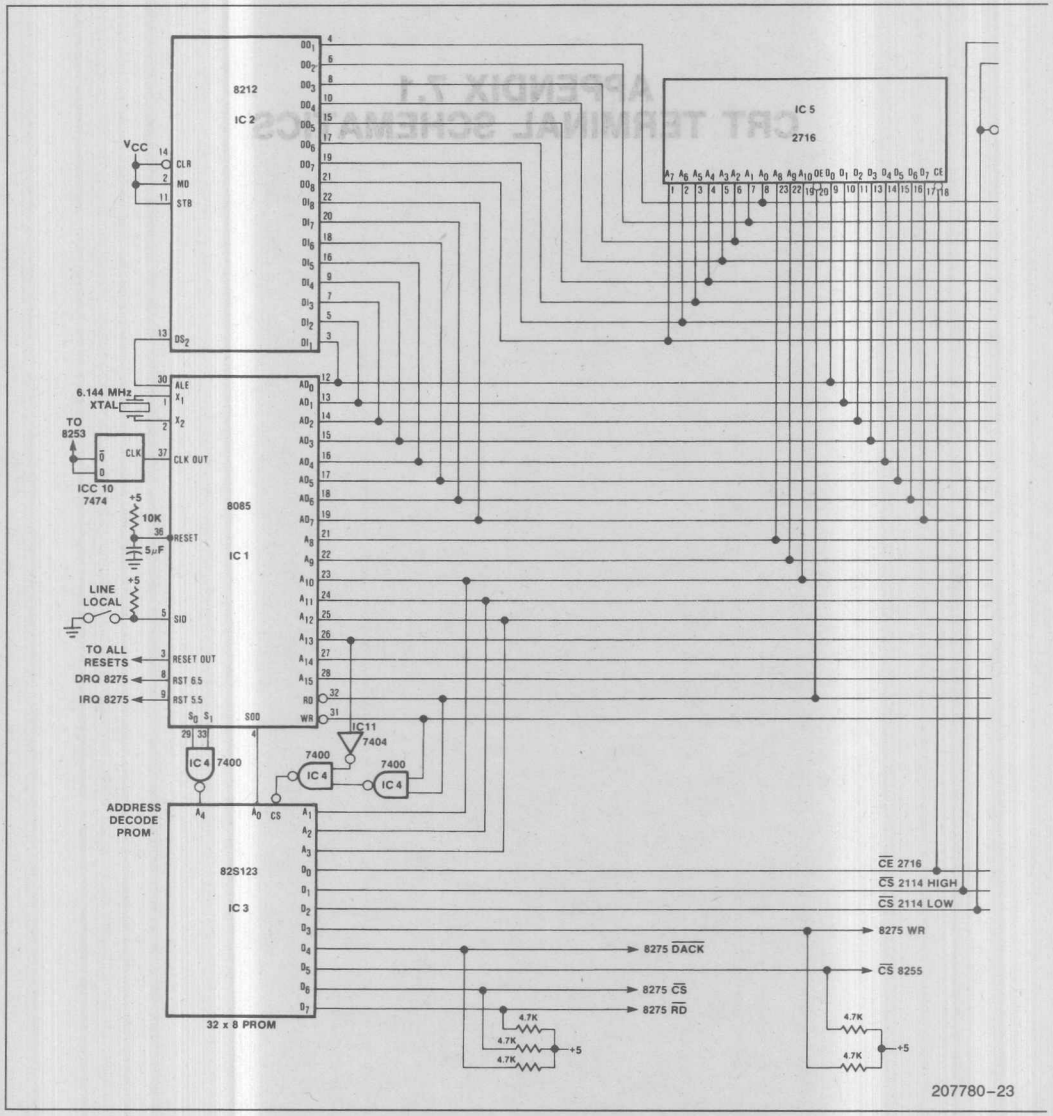


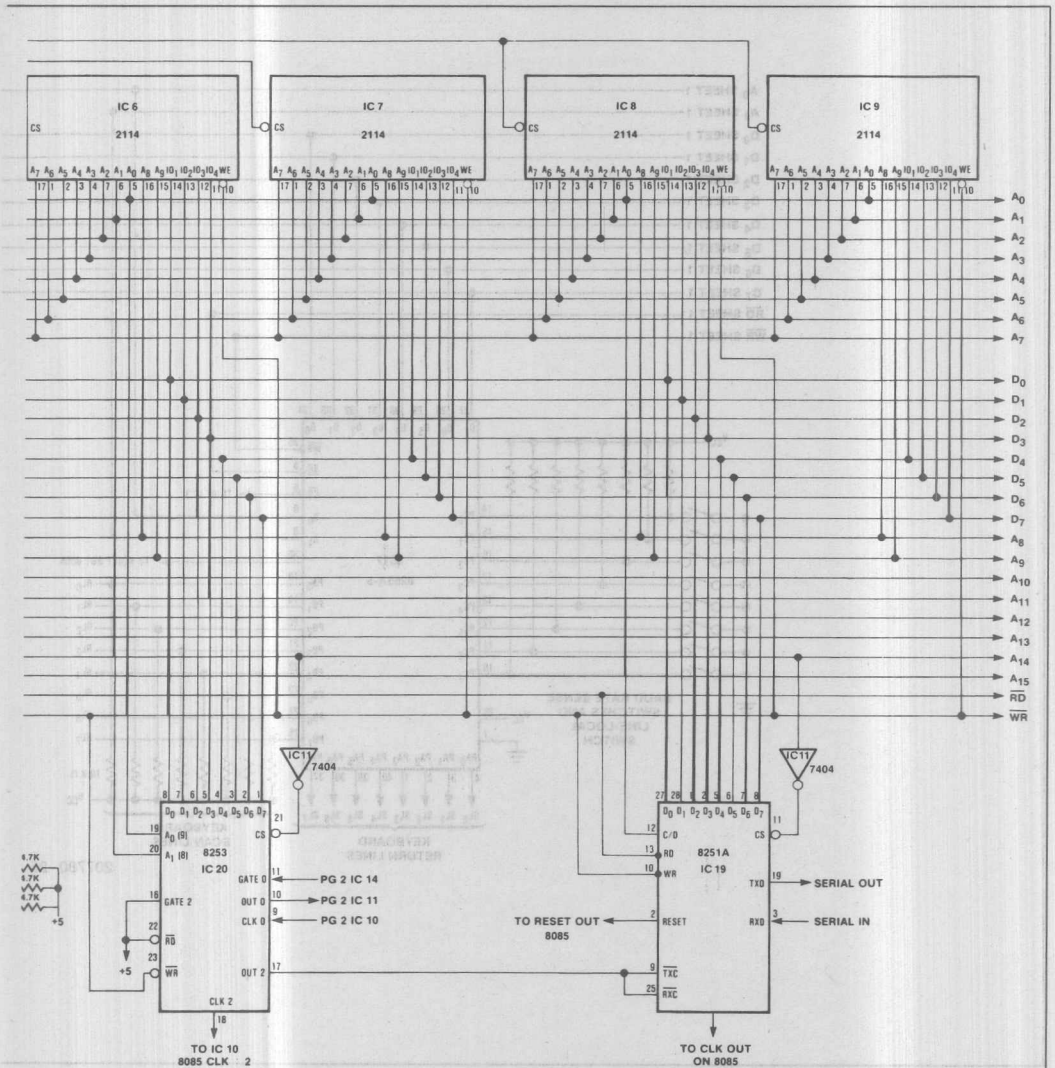
Figure 6-5. Timing Flowchart

Figure 6-5. Timing Flowchart

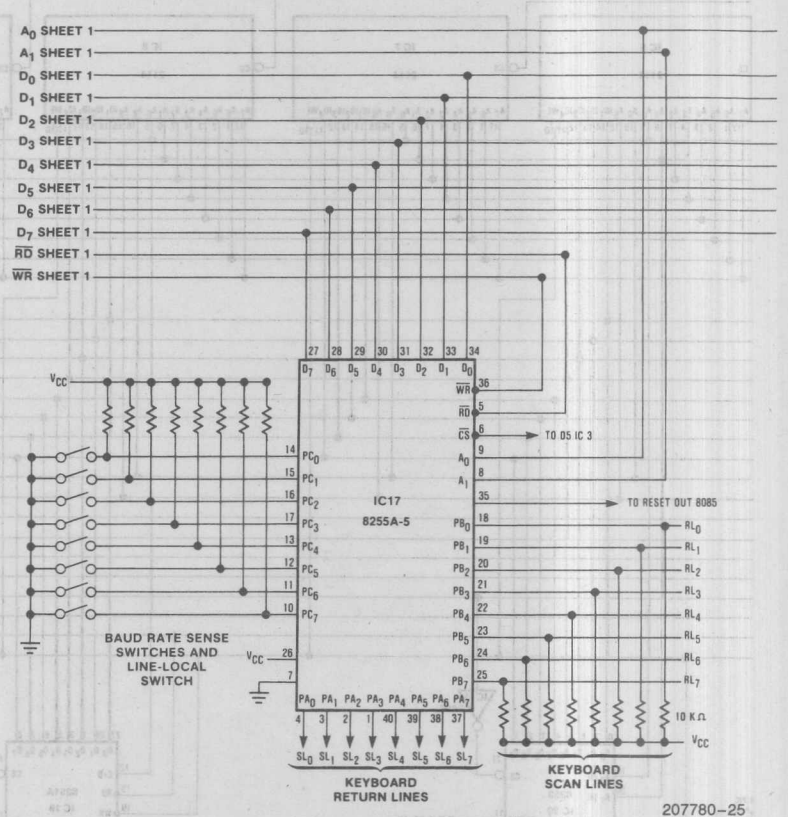
APPENDIX 7.1
CRT TERMINAL SCHEMATICS

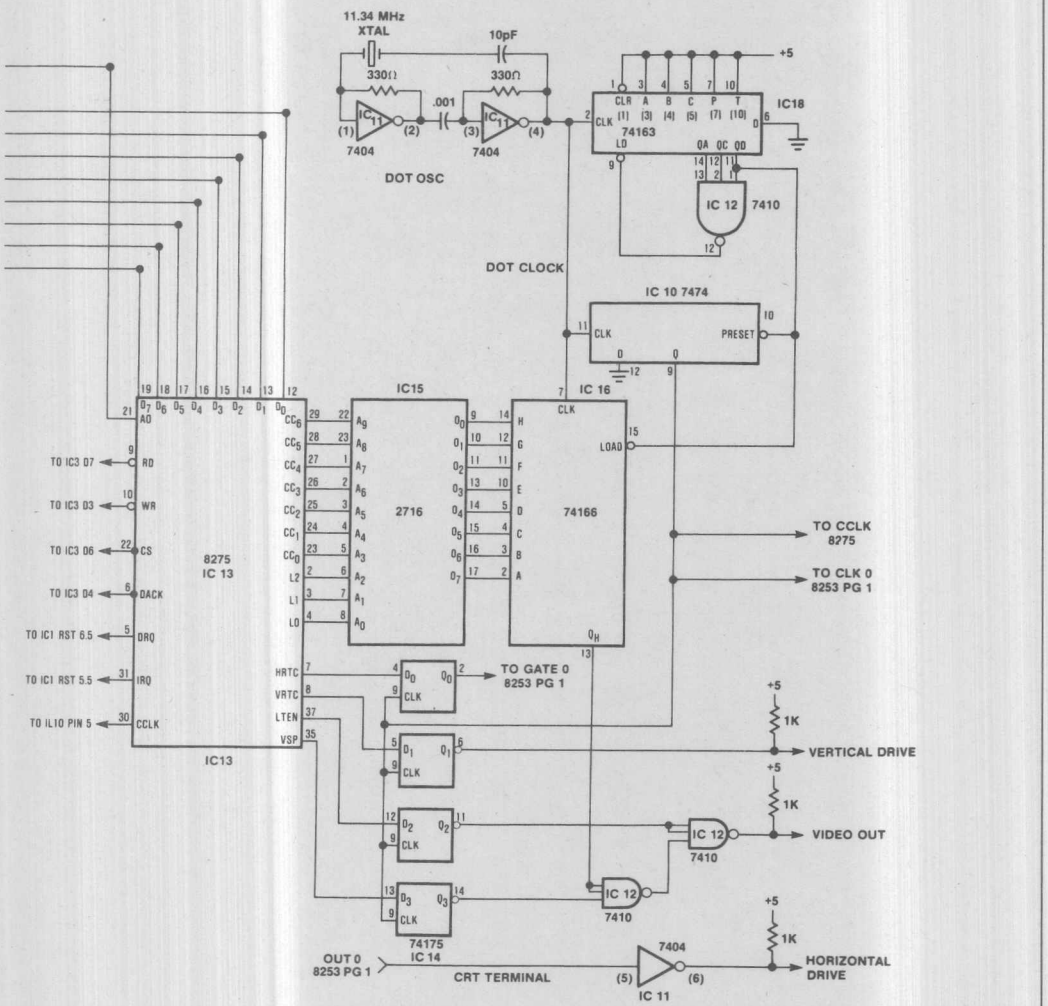



8212
IC 2
8085
IC 1
82S123
IC 3
32 x 8 PROM
IC 5
2716
6.144 MHz XTAL
TO 8253
ICC 10 7474
10K
5μF
LINE LOCAL
TO ALL RESETS
DRQ 8275
IRQ 8275
IC 4 7400
IC11 7404
ADDRESS DECODE PROM
CE 2716
CS 2114 HIGH
CS 2114 LOW
8275 WR
8275 DACK
CS 8255
8275 CS
8275 RD
4.7K
+5
207780-23


IC 6
IC 7
IC 8
IC 9
2114
CS
A0
A1
A2
A3
A4
A5
A6
A7
D0
D1
D2
D3
D4
D5
D6
D7
A8
A9
A10
A11
A12
A13
A14
A15
RD
WR
IC11
7404
8253
IC 20
GATE 0
OUT 0
CLK 0
GATE 2
OUT 2
CLK 2
PG 2 IC 14
PG 2 IC 11
PG 2 IC 10
4.7K
+5
TO IC 10
8085 CLK
8251A
IC 19
C/D
TXD
RXD
TXC
RXC
SERIAL OUT
SERIAL IN
TO RESET OUT
8085
TO CLK OUT
ON 8085

207780-24


A0 SHEET 1
A1 SHEET 1
D0 SHEET 1
D1 SHEET 1
D2 SHEET 1
D3 SHEET 1
D4 SHEET 1
D5 SHEET 1
D6 SHEET 1
D7 SHEET 1
RD SHEET 1
WR SHEET 1
IC17
8255A-5
TO D5 IC 3
TO RESET OUT 8085
BAUD RATE SENSE SWITCHES AND LINE-LOCAL SWITCH
KEYBOARD RETURN LINES
KEYBOARD SCAN LINES
10 KΩ
207780-25


11.34 MHz
XTAL
10pF
330Ω
330Ω
.001
IC 11
7404
DOT OSC
+5
IC18
74163
IC 12
7410
DOT CLOCK
IC 10 7474
PRESET
IC 16
IC15
2716
74166
8275
IC 13
TO IC3 D7
TO IC3 D3
TO IC3 D6
TO IC3 D4
TO IC1 RST 6.5
TO IC1 RST 5.5
TO IL10 PIN 5
TO CCLK
8275
TO CLK 0
8253 PG 1
TO GATE 0
8253 PG 1
VERTICAL DRIVE
VIDEO OUT
HORIZONTAL
DRIVE
1K
74175
IC 14
OUT 0
8253 PG 1
CRT TERMINAL
IC 11
207780-26

APPENDIX 7.2
KEYBOARD INTERFACE

The keyboard used in this design was a simple unencoded ASCII keyboard. In order to keep the cost to a minimum a simple scan matrix technique was implemented by using two ports of an 8255 parallel I/O device.

When the system is initialized the contents of the eight keyboard RAM locations are set to zero. Once every frame, which is 16.67 milliseconds the contents of the keyboard RAM is read and then rewritten with the contents of the current switch matrix. If a non-zero value of one of the keyboard RAM locations is found to be the same as the corresponding current switch matrix, a valid key push is registered and action is taken. By operating the keyboard scan in this manner an automatic debounce time of 16.67 milliseconds is provided.

Figure 7.2A shows the actual physical layout of the keyboard and Figure 7.2B shows how the individual keys were encoded. On Figure 7.2B the scan lines are the numbers on the bottom of each key position and the return lines are the numbers at the top of each key position. The shift, control, and caps lock key were brought in through separate lines of port C of the 8255. Figure 7.3 shows the basic keyboard matrix.

In order to guarantee that two scan lines could not be shorted together if two or more keys are pushed simultaneously, isolation diodes could be added as shown in Figure 7.4.

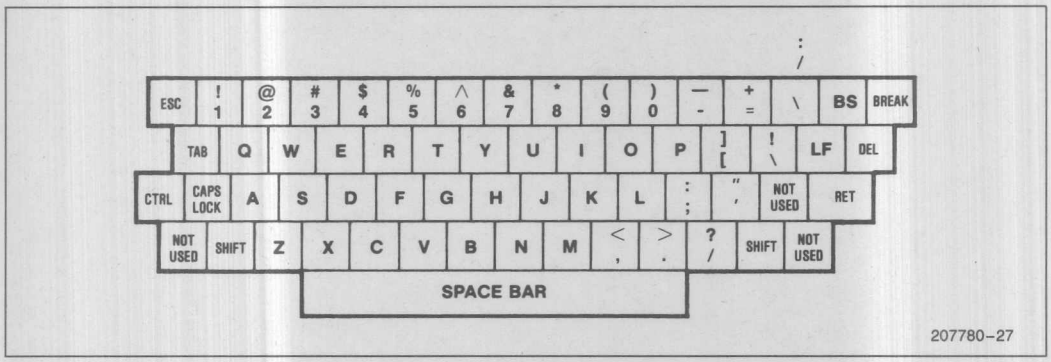


Figure 7-2A. Keyboard Layout

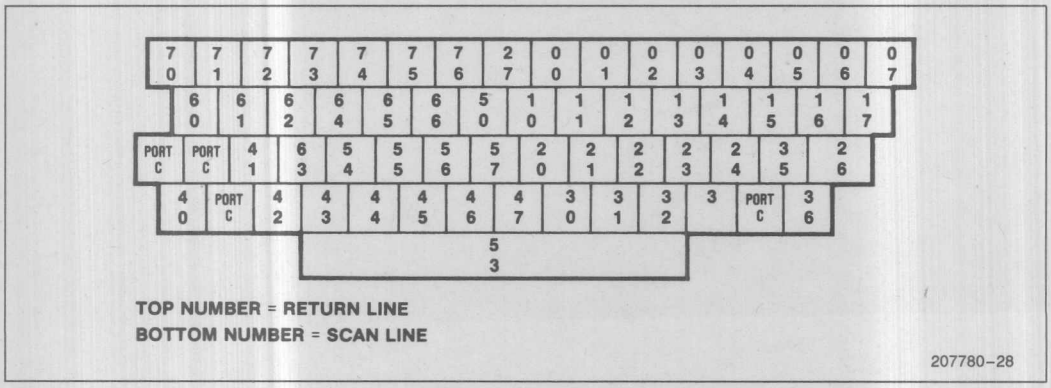


Figure 7-2B. Keyboard Encoding

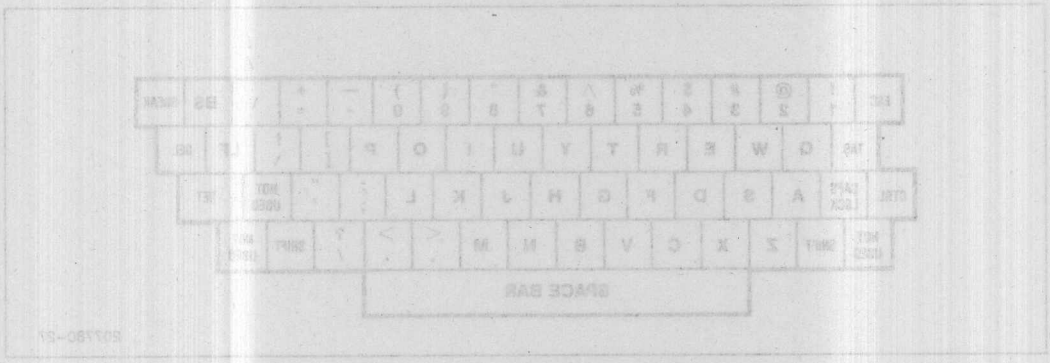


Figure 7-2A. Keyboard Layout

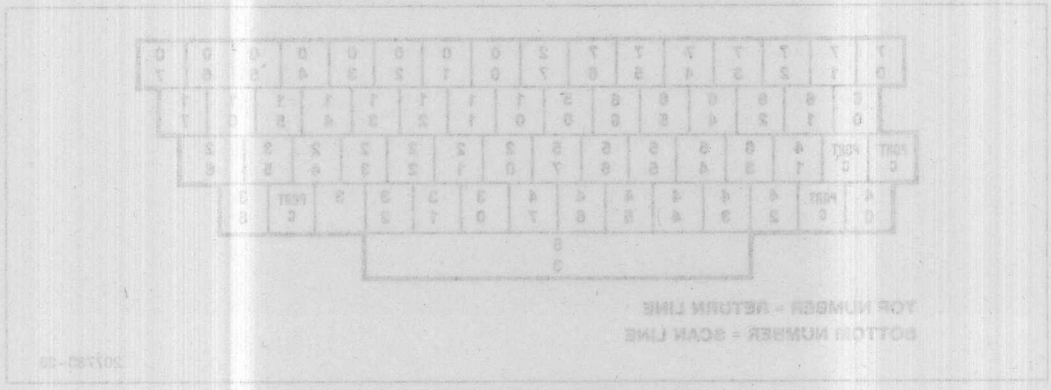


Figure 7-2B. Keyboard Encoding

APPENDIX 7.3
ESCAPE/CONTROL/DISPLAY CHARACTER SUMMARY

Bit	Control Characters		Displayable Character						Escape Sequence					
	000	001	010	011	100	101	110	111	010	011	100	101	110	111
0000	NUL @	DLE P	SP	ϕ	@	P		P						
0001	SOH A	DC1 Q	!	1	A	Q	A	Q			↑ A			
0010	STX B	DC2 R	"	2	B	R	B	R			↓ B			
0011	ETX C	DC3 S	#	3	C	S	C	S			→ C			
0100	EOT D	DC4 T	$	4	D	T	D	T			← D			
0101	ENQ E	NAK U	%	5	E	U	E	U			CLR E			
0110	ACK F	SYN V	&	6	F	V	F	V						
0111	BEL G	ETB W	'	7	G	W	G	W						
1000	BS H	CAN X	(	8	H	X	H	X			HOME H			
1001	HT I	EM Y	)	9	I	Y	I	Y						
1010	LF J	SUB Z	*	:	J	Z	J	Z			EOS I			
1011	VT K	ESC I	+	;	K	[	K				EL J			
1100	FF L	FS /	,	<	L	\	L							
1101	CR M	GS	-	=	M	]	M							
1110	SO N	RS ^	.	>	N	^	N							
1111	S1 O	US -	/	?	O	-	O							

NOTE:
Shaded blocks = functions terminal will react to. Others can be generated but are ignored upon receipt.

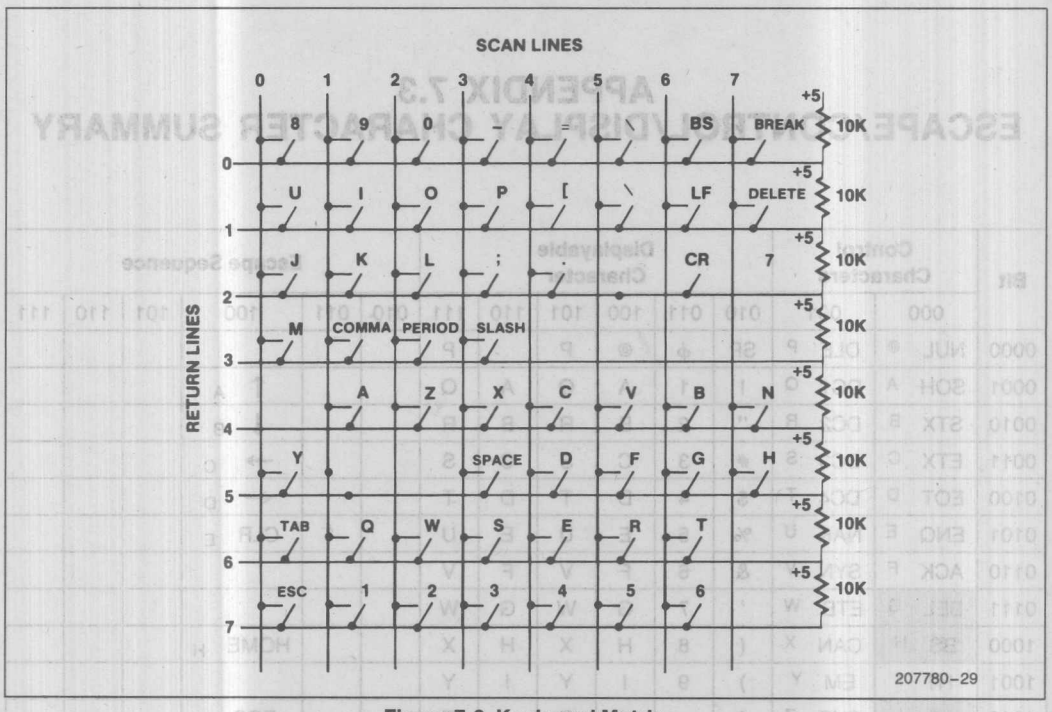


Figure 7-3. Keyboard Matrix

APPENDIX 7.4
PROM DECODING

As stated earlier, all of the logic necessary to convert the 8275 into a non-DMA type of device was performed by a single small bipolar prom. Besides turning certain processor READS into DACKS and WRITES for the 8275, this 32 by 8 prom decoded addresses for the system ram, rom, as well as for the 8255 parallel I/O port.

Any bipolar prom that has a by eight configuration could function in this application, This particular device was chosen simply because it is the only "by eight" prom available in a 16 pin package. The connection of the prom is shown in detail in Figure 7.5 and its truth table is shown in Figure 7.6. Note that when a fetch cycle (M1) is not being performed, the state of the SOD line is the only thing that determines if memory reads will be written into the 8275's row buffers. This is done by pulling both DACK and WRITE low on the 8275.

Also note that all of the outputs of the bipolar prom MUST BE PULLED HIGH by a resistor. This prevents any unwanted assertions when the prom is disabled.

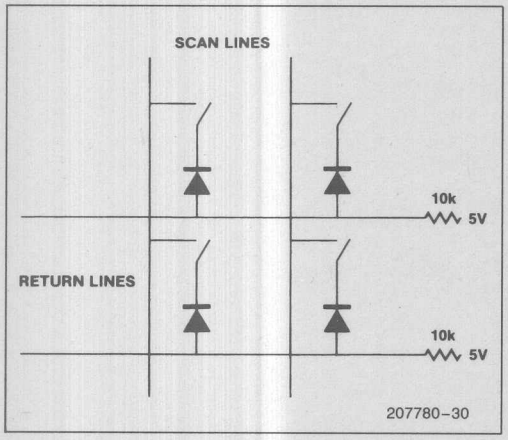


Figure 7-4. Isolating Scan Lines with Diodes

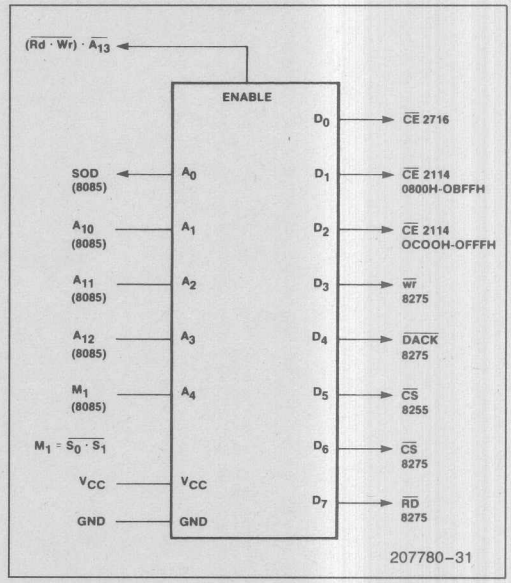


Figure 7-5. Bipolar Prom (825123) Connection

M1	A12	A11	A10	Sod	8275 RD	8275 CS	8255 CS	8275 DACK	8275 WR	2114 H	2114 L	2716
A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1	1	0
0	0	0	1	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	1	1	0
0	0	1	0	0	1	1	1	1	1	1	0	1
0	0	1	0	1	1	1	1	1	1	1	0	1
0	0	1	1	0	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	1	1	0	1	1
0	1	0	0	0	1	0	1	1	0	1	1	1
0	1	0	0	1	1	0	1	1	0	1	1	1
0	1	0	1	0	0	0	1	1	1	1	1	1
0	1	0	1	1	0	0	1	1	1	1	1	1
0	1	1	0	0	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	0	1	1	1	1	1
0	1	1	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	1	1	0	0	1	1	0
1	0	0	1	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	0	0	1	1	0
1	0	1	0	0	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	0	0	1	0	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	0	0	0	1	1
1	1	0	0	0	1	0	1	1	0	1	1	1
1	1	0	0	1	1	0	1	1	0	1	1	1
1	1	0	1	0	0	0	1	1	1	1	1	1
1	1	0	1	1	0	0	1	1	1	1	1	1
1	1	1	0	0	1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1

207780-32

Figure 7-6. Truth Table Bipolar Prom

APPENDIX 7.4
PROM DECODING

As stated earlier, all of the logic necessary to convert the 8275 into a non-DMA type of device was performed by a single small bipolar prom. Besides turning certain processor READS into DACKS and WRITES for the 8275, this 32 by 8 prom decoded addresses for the system ram, rom, as well as for the 8255 parallel I/O port.

Any bipolar prom that has a by eight configuration could function in this application. This particular device was chosen simply because it is the only "by eight" prom available in a 16 pin package. The connection of the prom is shown in detail in Figure 7.5 and its truth table is shown in Figure 7.6. Note that when a fetch cycle (M1) is not being performed, the state of the SOD line is the only thing that determines if memory reads will be written into the 8275's row buffers. This is done by pulling both DACK and WRITE low on the 8275.

Also note that all of the outputs of the bipolar prom MUST BE PULLED HIGH by a resistor. This prevents any unwanted assertions when the prom is disabled.

Figure 7-5. Bipolar Prom (82S123) Connection

Figure 7-6. Truth Table Bipolar Prom

Figure 7-4. Isolating Scan Lines with Diodes

APPENDIX 7.5
CHARACTER GENERATOR

As previously mentioned, the character generator used in this terminal is a 2716 or 2758 EPROM. A 1K by 8 device is sufficient since a 128 character 5 by 7 dot matrix only required 8K of memory. Any "standard" or custom character generator could have been used.

The three low-order line count outputs (LC0–LC2) from the 8275 are connected to the three low-order address lines of the character generator and the seven character generator outputs (CC0–CC6) are connected to the A3–A9 of the character generator. The output from the character generator is loaded into a shift register and the serial output from the shift register is the video output of the terminal.

Now, let's assume that the letter "E" is to be displayed. The ASCII code for "E" is 45H. So, 45H is presented to address lines A2–A9 of the character generator. The scan lines will now count each line from zero to seven to "form" the character as shown in Figure 7.7. This same procedure is used to form all 128 possible characters.

It should be obvious that "custom" character fonts could be made just by changing the bit patterns in the character generator PROM. For reference, Appendix 7.6 contains a HEX dump of the character generator used in this terminal.

45H = 01000101

Address to Prom = 01000101 SL2 SL1 SL0

= 228H – 22FH

Depending on state of Scan lines.

Character generator output

ROM Address	Rom Hex Output	Bit Output*						
		0	1	2	3	4	5	6	7
228H	3E								
229H	02		X	X	X	X	X		
22AH	02		X						
22BH	0E		X						
22CH	02		X	X	X				
22DH	02		X						
22EH	3E		X						
22FH	00		X	X	X	X	X		

Bits 0, 6, and 7 are not used.

*NOTE:
Bit output is backward from convention.

Figure 7.7. Character Generation

APPENDIX 7.5
CHARACTER GENERATOR

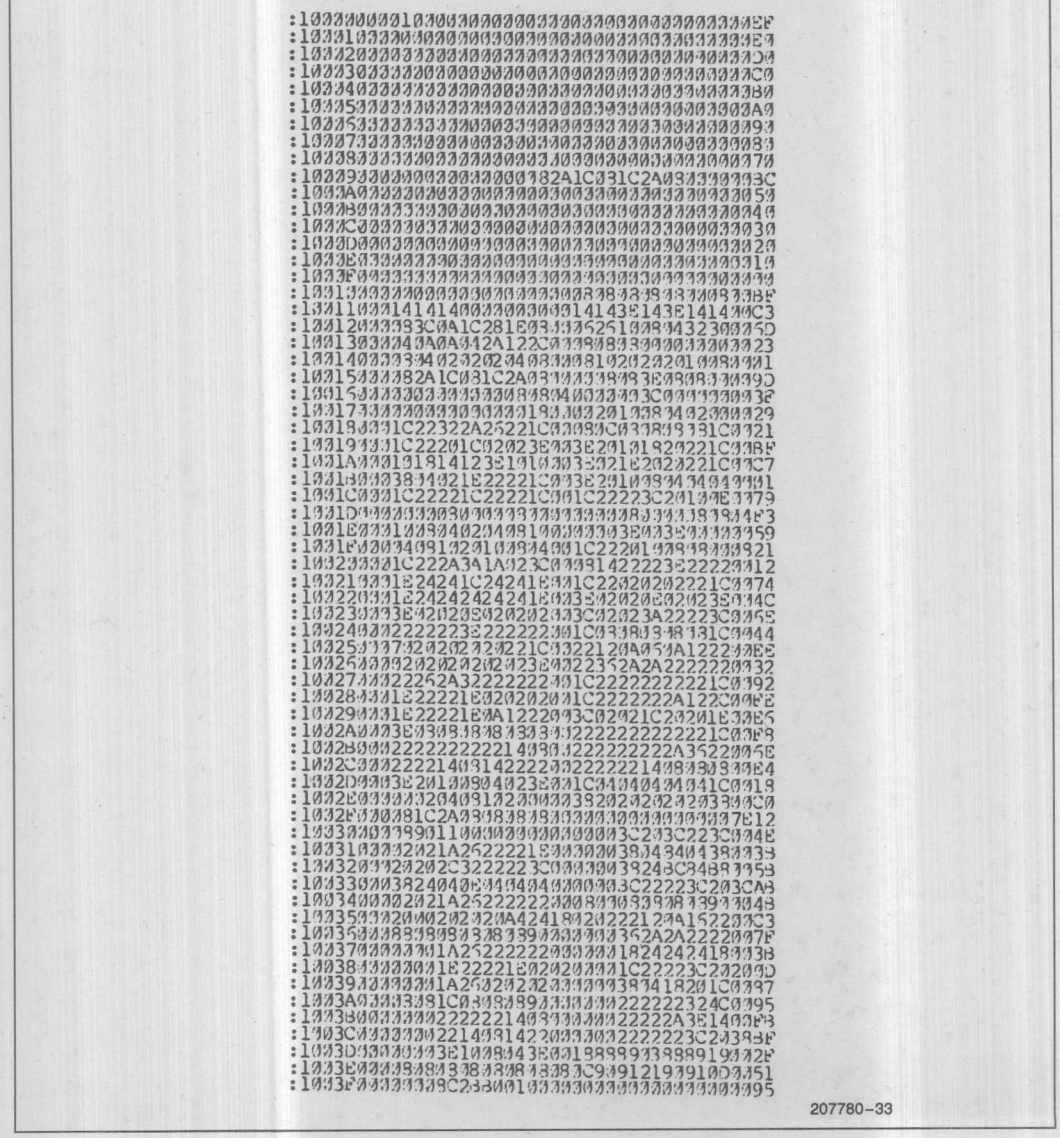
As previously mentioned, the character generator used in this terminal is a 2716 or 2758 EPROM. A 1K by 8 device is sufficient since a 128 character 5 by 7 dot matrix only required 5K of memory. Any "standard" or custom character generator could have been used.

The three low-order line count outputs (LC0–LC2) from the 8275 are connected to the three low-order address lines of the character generator and the seven character generator outputs (CC0–CC6) are connected to the A3–A9 of the character generator. The output from the character generator is loaded into a shift register and the serial output from the shift register is the video output of the terminal.

Now, let's assume that the letter "E" is to be displayed. The ASCII code for "E" is 45H. So, 45H is presented to address lines A2–A9 of the character generator. The seven lines will now count each line from zero to seven to "form" the character as shown in Figure 7.7. This same procedure is used to form all 128 possible characters.

It should be obvious that "custom" character fonts could be made just by changing the bit patterns in the character generator PROM. For reference, Appendix 7.6 contains a HEX dump of the character generator used in this terminal.

APPENDIX 7.6
HEX DUMP OF CHARACTER GENERATOR



207780-33

APPENDIX 7.6
HEX DUMP OF CHARACTER GENERATOR

APPENDIX 7.7
COMPOSITE VIDEO

In this design, it was assumed that the monitor required a separate horizontal drive, vertical drive, and video input. However, many monitors require a composite video signal. The schematic shown in Figure 7.8 illustrates how to generate a composite video signal from the output of the 8275.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the vertical and horizontal timing is used to "center" the display. VR1 and VR2 control the amount of delay. IC3 is used to mix the vertical and horizontal retrace and Q1 along with the R1, R2, and R3 mix the video and the retrace signal and provide the proper DC levels.

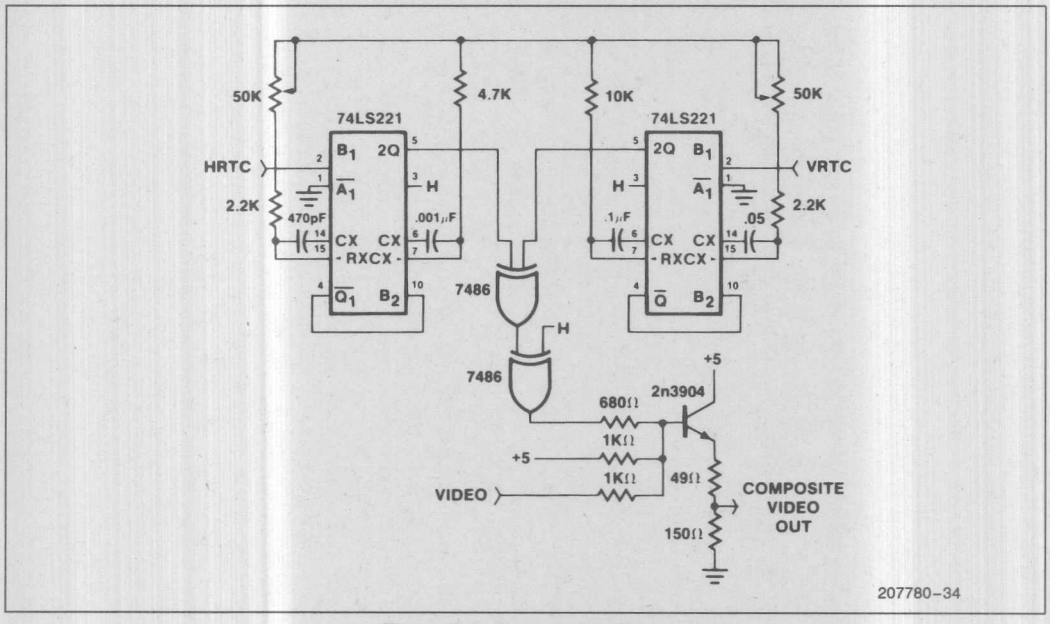


Figure 7-8. Composite Video

APPENDIX 7.7
COMPOSITE VIDEO

In this design, it was assumed that the monitor required a separate horizontal drive, vertical drive, and video input. However, many monitors require a composite video signal. The schematic shown in Figure 7.8 illustrates how to generate a composite video signal from the output of the 8275.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the vertical and horizontal timing is used to "center" the display. VR1 and VR2 control the amount of delay. IC3 is used to mix the vertical and horizontal retrace and Q1 along with the R1, R2, and R3 mix the video and the retrace signal and provide the proper DC levels.

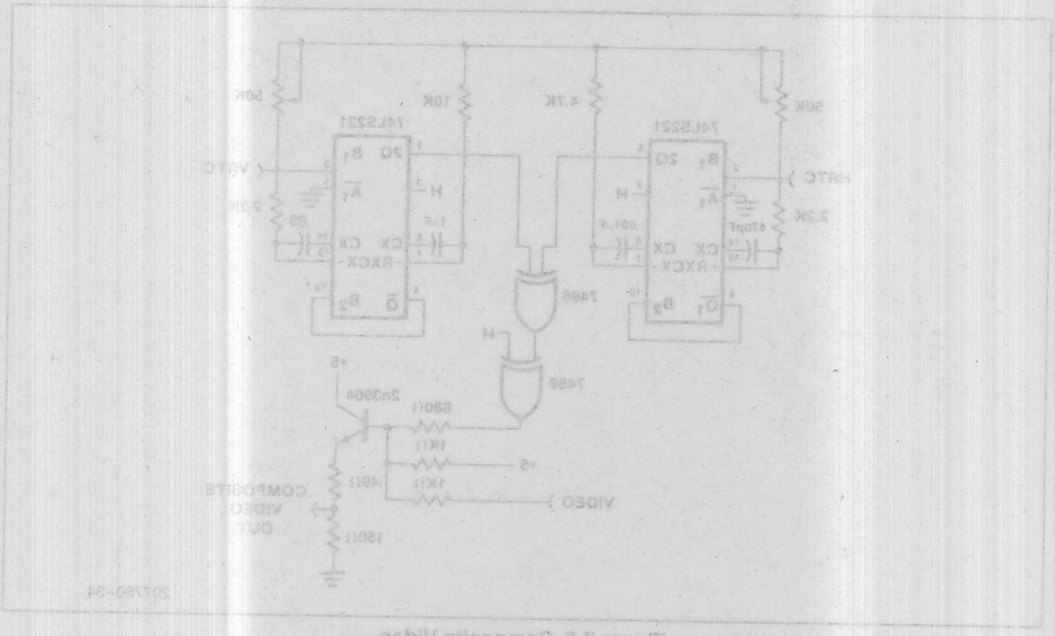


Figure 7-8. Composite Video

APPENDIX 7.8
SOFTWARE LISTINGS

```
ISIS-II 8080/8085 MACRO ASSEMBLER, X108

LOC  OBJ          SEQ        SOURCE STATEMENT

                     1 $MOD85  MACROFILE
                     2         ;NO DMA 8275 SOFTWARE ALL I/O IS MEMORY MAPPED
                     3         ;SYSTEM ROM 0000H TO 07FFH
                     4         ;SYSTEM RAM 0800H TO 0FFFH
                     5         ;8275 WRITE 1000H TO 13FFH
                     6         ;8275 READ 1400H TO 17FFH
                     7         ;8255 READ/WRITE 1800H TO 1FFF
                     8         ;8253 ENABLED BY A14
                     9         ;8251 ENABLED BY A15
1800                10 PORTA    EQU     1800H           ;8255 PORT A ADDRESS
1801                11 PORTB    EQU     1801H           ;8255 PORT B ADDRESS
1802                12 PORTC    EQU     1802H           ;8255 PORT C ADDRESS
1803                13 CNWD55   EQU     1803H           ;8255 CONTROL PORT ADDRESS
A001                14 USTF     EQU     0A001H          ;8251 FLAGS
A000                15 USTD     EQU     0A000H          ;8251 DATA
6000                16 CNT0     EQU     6000H           ;8253 COUNTER 0
6001                17 CNT1     EQU     6001H           ;8253 COUNTER 1
6002                18 CNT2     EQU     6002H           ;8253 COUNTER 2
6003                19 CNTM     EQU     6003H           ;8253 MODE WORD
1001                20 CRTS     EQU     1001H           ;8275 CONTROL ADDRESS
1000                21 CRTM     EQU     1000H           ;8275 MODE ADDRESS
1401                22 INT75    EQU     1401H           ;8275 INTERRUPT CLEAR
0800                23 TPDIS    EQU     0800H           ;TOP OF DISPLAY RAM
0F80                24 BTDIS    EQU     0F80H           ;BOTTOM OF DISPLAY RAM
0FD0                25 LAST     EQU     0FD0H           ;FIRST BYTE AFTER DISPLAY
0018                26 CURBOT   EQU     18H             ;BOTTOM Y CURSOR
0050                27 LNGTH    EQU     0050H           ;LENGTH OF ONE LINE
0FE0                28 STPTR    EQU     0FE0H           ;LOCATION OF STACK POINTER
                    29         ;
                    30         ;START PROGRAM
                    31         ;ALL VARIABLES ARE INITIALIZED BEFORE ANYTHING ELSE
                    32         ;
0000 F3             33         DI                      ;DISABLE INTERRUPTS
0001 31E00F         34         LXI     SP,STPTR        ;LOAD STACK POINTER
0004 210008         35         LXI     H,TPDIS         ;LOAD H&L WITH TOP OF DISPLAY
0007 22E30F         36         SHLD    TOPAD           ;SET TOP = TOP OF DISPLAY
000A 22E80F         37         SHLD    CURAD           ;STORE THE CURRENT ADDRESS
000D 3E00           38         MVI     A,00H           ;ZERO A
000F 32E10F         39         STA     CURSY           ;ZERO CURSOR Y POINTER
0012 32E20F         40         STA     CURSX           ;ZERO CURSOR X POINTER
0015 32E80F         41         STA     KBCHR           ;ZERO KBD CHARACTER
0018 32E70F         42         STA     USCHR           ;ZERO USART CHAR BUFFER
001B 32EA0F         43         STA     KEYDWN          ;ZERO KEY DOWN
```

207780-35

```
001E 32ED0F        44          STA     KEYOK           ;ZERO KEYOK
0021 32EE0F        45          STA     ESCP            ;ZERO ESCAPE
0024 C39800        46          JMP     LPKBD           ;JUMP AND SET EVERYTHING UP
                   47          ;
                   48          ;THIS JUMP VECTOR IS LOCATED AT THE RST 5.5 LOCATION
                   49          ;OF THE 8085. IT IS USED TO READ THE 8275 STATUS AND
                   50          ;READ THE KEYBOARD. THIS ROUTINE IS EXECUTED ONCE EVERY
                   51          ;16.667 MILLISECONDS.
                   52          ;
002C               53          ORG     002CH
002C C36701        54          JMP     FRAME
                   55          ;
                   56          ;THIS ROUTINE IS LOCATED AT THE RST 6.5 LOCATION OF THE
                   57          ;8085 AND IS USED TO LOAD THE DATA TO BE DISPLAYED INTO
                   58          ;THE 8275. THIS ROUTINE IS EXECUTED ONCE EVERY 617 MICROSECONDS.
                   59          ;
0034               60          ORG     34H
0034 F5            61 POPDAT:  PUSH    PSW             ;SAVE A AND FLAGS
0035 E5            62          PUSH    H               ;SAVE H AND L
0036 D5            63          PUSH    D               ;SAVE D AND E
0037 210000        64          LXI     H,0000H         ;ZERO H AND L
003A 39            65          DAD     SP              ;PUT STACK POINTER IN H AND L
003B EB            66          XCHG                    ;PUT STACK IN D AND E
003C 2AE80F        67          LHLD    CURAD           ;GET POINTER
003F F9            68          SPHL                    ;PUT CURRENT LINE INTO SP
0040 3EC0          69          MVI     A,0C0H          ;SET MASK FOR SIM
0042 30            70          SIM
                   71          REPT (LNGTH/2)
                   72          POP     H
                   73          ENDM
0043 E1            74+         POP     H
0044 E1            75+         POP     H
0045 E1            76+         POP     H
0046 E1            77+         POP     H
0047 E1            78+         POP     H
0048 E1            79+         POP     H
0049 E1            80+         POP     H
004A E1            81+         POP     H
004B E1            82+         POP     H
004C E1            83+         POP     H
004D E1            84+         POP     H
004E E1            85+         POP     H
004F E1            86+         POP     H
0050 E1            87+         POP     H
0051 E1            88+         POP     H
0052 E1            89+         POP     H
0053 E1            90+         POP     H
0054 E1            91+         POP     H
0055 E1            92+         POP     H
0056 E1            93+         POP     H
0057 E1            94+         POP     H
0058 E1            95+         POP     H
0059 E1            96+         POP     H
005A E1            97+         POP     H
0058 E1            98+         POP     H
005C E1            99+         POP     H
005D E1           100+         POP     H
005E E1           101+         POP     H
005F E1           102+         POP     H
0060 E1           103+         POP     H
0061 E1           104+         POP     H
0062 E1           105+         POP     H
0063 E1           106+         POP     H
0064 E1           107+         POP     H
0065 E1           108+         POP     H
0066 E1           109+         POP     H
0067 E1           110+         POP     H
0068 E1           111+         POP     H
0069 E1           112+         POP     H
006A E1           113+         POP     H
006B 0F           114          RRC                     ;SET UP A
006C 30           115          SIM                     ;GO BACK TO NORMAL MODE
006D 210000       116          LXI     H,0000H         ;ZERO HL
0070 39           117          DAD     SP              ;ADD STACK
0071 EB           118          XCHG                    ;PUT STACK IN H AND L
0072 F9           119          SPHL                    ;RESTORE STACK
0073 21D00F       120          LXI     H,LAST          ;PUT BOTTOM DISPLAY IN H AND L
0076 EB           121          XCHG                    ;SWAP REGISTERS
0077 7A           122          MOV     A,D             ;PUT HIGH ORDER IN A
0078 BC           123          CMP     H               ;SEE IF SAME AS H
0079 C28400       124          JNZ     KPTK            ;IF NOT LEAVE
007C 7B           125          MOV     A,E             ;PUT LOW ORDER IN A
007D BD           126          CMP     L               ;SEE IF SAME AS L
007E C28400       127          JNZ     KPTK            ;IF NOT LEAVE
0081 210008       128          LXI     H,TPDIS         ;LOAD H AND L WITH TOP OF SCREEN MEMORY
0084 22E80F       129 KPTK:   SHLD    CURAD           ;PUT BACK CURRENT ADDRESS
0087 3E18         130          MVI     A,18H           ;SET MASK
0089 30           131          SIM                     ;OUTPUT MASK
```

207780-36

```
008A D1          132          POP     D               ;GET D AND E
008B E1          133          POP     H               ;GET H AND L
008C F1          134          POP     PSW             ;GET A AND FLAGS
008D FB          135          EI                      ;TURN ON INTERRUPTS
008E C9          136          RET                     ;GO BACK
                 137          ;
                 138          ;THIS IS THE EXIT ROUTINE FOR THE FRAME INTERRUPT
                 139          ;
008F 3E18        140 BYPASS:  MVI     A,18H           ;SET MASK
0091 30          141          SIM                     ;OUTPUT THE MASK
0092 C1          142          POP     B               ;GET B AND C
0093 D1          143          POP     D               ;GET D AND E
0094 E1          144          POP     H               ;GET H AND L
0095 F1          145          POP     PSW             ;GET A AND FLAGS
0096 FB          146          EI                      ;ENABLE INTERRUPTS
0097 C9          147          RET                     ;GO BACK
                 148          ;
                 149          ;THIS CLEARS THE AREA OF RAM THAT IS USED
                 150          ;FOR KEYBOARD DEBOUNCE.
                 151          ;
0098 32EF0F      152 LPK8D:   STA     SHCON           ;ZERO SHIFT CONTROL
009B 32F00F      153          STA     RETLIN          ;ZERO RETURN LINE
009E 32F10F      154          STA     SCNLIN          ;ZERO SCAN LINE
                 155          ;
                 156          ;THIS ROUTINE CLEARS THE ENTIRE SCREEN BY PUTTING
                 157          ;SPACE CODES (20H) IN EVERY LOCATION ON THE SCREEN.
                 158          ;
00A1 210008      159          LXI     H,TPDIS         ;PUT TOP OF SCREEN IN HL
00A4 01D00F      160          LXI     B,LAST          ;PUT BOTTOM IN BC
00A7 3620        161 LOOPF:   MVI     M,20H           ;PUT SPACE IN M
00A9 23          162          INX     H               ;INCREMENT POINTER
00AA 7C          163          MOV     A,H             ;GET H
00AB B8          164          CMP     B               ;SEE IF SAME AS B
00AC C2A700      165          JNZ     LOOPF           ;IF NOT LOOP AGAIN
00AF 7D          166          MOV     A,L             ;GET L
00B0 B9          167          CMP     C               ;SEE IF SAME AS C
00B1 C2A700      168          JNZ     LOOPF           ;IF NOT LOOP AGAIN
                 169          ;
                 170          ;8255 INITIALIZATION
                 171          ;
00B4 3E88        172          MVI     A,88H           ;MOVE 8255 CONTROL WORD INTO A
00B6 320318      173          STA     CNWD55          ;PUT CONTROL WORD INTO 8255
                 174          ;
                 175          ;8251 INITIALIZATION
                 176          ;
00B9 2101A0      177          LXI     H,USTF          ;GET 8251 FLAG ADDRESS
00BC 3680        178          MVI     M,80H           ;DUMMY STORE TO 8251
00BE 3600        179          MVI     M,00H           ;RESET 8251
00C0 3640        180          MVI     M,40H           ;RESET 8251
00C2 00          181          NOP                     ;WAIT
00C3 36EA        182          MVI     M,0EAH          ;LOAD 8251 MODE WORD
00C5 3605        183          MVI     M,05H           ;LOAD 8251 COMMAND WORD
                 184          ;
                 185          ;8253 INITIALIZATION
                 186          ;
00C7 3E32        187          MVI     A,32H           ;CONTROL WORD FOR 8253
00C9 320360      188          STA     CNTM            ;PUT CONTROL WORD INTO 8253
00CC 3E32        189          MVI     A,32H           ;LSB 8253
00CE 320060      190          STA     CNT0            ;PUT IT IN 8235
00D1 3E00        191          MVI     A,00H           ;MSB 8253
00D3 320060      192          STA     CNT0            ;PUT IT IN 8253
00D6 CDDC00      193          CALL    STBAUD          ;GO DO BAUD RATE
00D9 C3F900      194          JMP     IN75            ;GO DO 8275
                 195          ;
                 196          ;THIS ROUTINE READS THE BAUD RATE SWITCHES FROM PORT C
                 197          ;OF THE 8255 AND LOOKS UP THE NUMBERS NEEDED TO LOAD
                 198          ;THE 8253 TO PROVIDE THE PROPER BAUD RATE.
                 199          ;
00DC 3A0218      200 STBAUD:  LDA     PORTC           ;READ BAUD RATE SWITCHES
00DF E60F        201          ANI     0FH             ;STRIP OFF 4 MSB'S
00E1 32EC0F      202          STA     BAUD            ;SAVE IT
00E4 07          203          RLC                     ;MOVE BITS OVER ONE PLACE
00E5 21C505      204          LXI     H,BDLK          ;GET BAUD RATE LOOK UP TABLE
00E8 1600        205          MVI     D,00H           ;ZERO D
00EA 5F          206          MOV     E,A             ;PUT A IN E
00EB 19          207          DAD     D               ;GET OFFSET
00EC 110360      208          LXI     D,CNTM          ;POINT DE TO 8253
00EF 3E86        209          MVI     A,086H          ;GET CONTROL WORD
00F1 12          210          STAX    D               ;STORE IN 8253
00F2 1B          211          DCX     D               ;POINT AT #2 COUNTER
00F3 7E          212          MOV     A,M             ;GET LSB BAUD RATE
00F4 12          213          STAX    D               ;PUT IT IN 8253
00F5 23          214          INX     H               ;POINT AT MSB BAUD RATE
00F6 7E          215          MOV     A,M             ;GET MSB BAUD RATE
00F7 12          216          STAX    D               ;PUT IT IN 8253
00F8 C9          217          RET                     ;GO BACK
                 218          ;
```

207780-37

```
                        ;8275 INITIALIZATION
                        ;
00F9 210110     IN75:   LXI     H,CRTS
00FC 3600               MVI     M,00H           ;RESET AND STOP DISPLAY
00FE 2B                 DCX     H               ;HL=1000H
00FF 364F               MVI     M,4FH           ;SCREEN PARAMETER BYTE 1
0101 3658               MVI     M,58H           ;SCREEN PARAMETER BYTE 2
0103 3689               MVI     M,89H           ;SCREEN PARAMETER BYTE 3
0105 36DD               MVI     M,0DDH          ;SCREEN PARAMETER BYTE 4
0107 23                 INX     H               ;HL=1001H
0108 CD8803             CALL    LDCUR           ;LOAD THE CURSOR
010B 3E00               MVI     M,0E0H          ;PRESET COUNTERS
010D 3623               MVI     M,23H           ;START DISPLAY
                        ;
                        ;THIS ROUTINE READS BOTH THE KEYBOARD AND THE USART
                        ;AND TAKES PROPER ACTION DEPENDING ON HOW THE LINE-LOCAL
                        ;SWITCH IS SET
                        ;
010F 3E18       SETUP:  MVI     A,18H           ;SET MASK
0111 30                 SIM                     ;LOAD MASK
0112 FB                 EI                      ;ENABLE INTERRUPTS
                        ;
                        ;READ THE USART
                        ;
0113 20         RXRDY:  RIM                     ;GET LINE LOCAL
0114 E680               ANI     80H             ;IS IT ON OR OFF?
0116 C22101             JNZ     KEYINP          ;LEAVE IF IT IS ON
0119 3A01A0             LDA     USTF            ;READ 8251 FLAGS
011C E602               ANI     02H             ;LOOK AT RXRDY
011E C25C01             JNZ     OK7             ;IF HAVE CHARACTER GO TO WORK
0121 3AEA0F     KEYINP: LDA     KEYDWN          ;GET KEYBOARD CHARACTER
0124 E680               ANI     80H             ;IS IT THERE
0126 C23101             JNZ     KEYS            ;IF KEY IS PUSHED LEAVE
0129 3E00               MVI     A,00H           ;ZERO A
012B 32ED0F             STA     KEYOK           ;CLEAR KEYOK
012E C31301             JMP     RXRDY           ;LOOP AGAIN
0131 3AED0F     KEYS:   LDA     KEYOK           ;WAS KEY DOWN
0134 4F                 MOV     C,A             ;SAVE A IN C
0135 3AEB0F             LDA     KBCHR           ;GET KEYBOARD CHARACTER
0138 B9                 CMP     C               ;IS IT THE SAME AS KEYOK
0139 CA1301             JZ      RXRDY           ;IF SAME LOOP AGAIN
013C 32ED0F             STA     KEYOK           ;IF NOT SAVE IT
013F 32E70F             STA     USCHR           ;SAVE IT
0142 20                 RIM                     ;GET LINE LOCAL
0143 E680               ANI     80H             ;WHICH WAY
0145 CA4B01             JZ      TRANS           ;LEAVE IF LINE
0148 C34E02             JMP     CHREC           ;TIME TO DO SOME WORK
014B 3A01A0     TRANS:  LDA     USTF            ;GET USART FLAGS
014E E601               ANI     01H             ;READY TO TRANSMIT?
0150 CA4B01             JZ      TRANS           ;LOOP IF NOT READY
0153 3AE70F             LDA     USCHR           ;GET CHARACTER
0156 3200A0             STA     USTD            ;PUT IN USART
0159 C30F01             JMP     SETUP           ;LEAVE
015C 3A00A0     OK7:    LDA     USTD            ;READ USART
015F E67F               ANI     07FH            ;STRIP MSB
0161 32E70F             STA     USCHR           ;PUT IT IN MEMORY
0164 C34E02             JMP     CHREC           ;LEAVE
                        ;
                        ;THIS ROUTINE CHECKS THE BAUD RATE SWITCHES, RESETS THE
                        ;SCREEN POINTERS AND READS AND LOOKS UP THE KEYBOARD.
                        ;
0167 F5         FRAME:  PUSH    PSW             ;SAVE A AND FLAGS
0168 E5                 PUSH    H               ;SAVE H AND L
0169 D5                 PUSH    D               ;SAVE D AND E
016A C5                 PUSH    B               ;SAVE B AND C
016B 3A0114             LDA     INT75           ;READ 8275 TO CLEAR INTERRUPT
                        ;
                        ;SET UP THE POINTERS
                        ;
016E 2AE30F             LHLD    TOPAD           ;LOAD TOP IN H AND L
0171 22E80F             SHLD    CURAD           ;STORE TOP IN CURRENT ADDRESS
                        ;
                        ;SET UP BAUD RATE
                        ;
0174 3A0218             LDA     PORTC           ;READ BAUD RATE SWITCHES
0177 E60F               ANI     0FH             ;STRIP OFF 4 MSB'S
0179 47                 MOV     B,A             ;SAVE IN B
017A 3AEC0F             LDA     BAUD            ;GET BAUD RATE
017D B8                 CMP     B               ;SEE IF SAME AS B
017E C4DC00             CNZ     STBAUD          ;IF NOT SAME DO SOMETHING
                        ;
                        ;READ KEYBOARD
                        ;
0181 3AEA0F             LDA     KEYDWN          ;SEE IF A KEY IS DOWN
0184 E540               ANI     40H             ;SET THE FLAGS
0186 C2C201             JNZ     KYDOWN          ;IF KEY IS DOWN JUMP AROUND
0189 CD8F01             CALL    RDKB            ;GO READ THE KEYBOARD
018C C38F00             JMP     BYPASS          ;LEAVE
```

207780-38

```
018F 21EF0F      307 RDKB:   LXI     H,SHCON     ;POINT HL AT KEYBOARD RAM
0192 3A0218      308         LDA     PORTC       ;GET CONTROL AND SHIFT
0195 77          309         MOV     M,A         ;SAVE IN MEMORY
0196 3EFE        310         MVI     A,0FEH      ;SET UP A
0198 320018      311 LOOPK:  STA     PORTA       ;OUTPUT A
019B 47          312         MOV     B,A         ;SAVE A IN B
019C 3A0118      313         LDA     PORTB       ;READ KEYBOARD
019F 2F          314         CMA                 ;INVERT A
01A0 B7          315         ORA     A           ;SET THE FLAGS
01A1 C2AF01      316         JNZ     SAVKEY      ;LEAVE IF KEY IS DOWN
01A4 78          317         MOV     A,B         ;GET SCAN LINE BACK
01A5 07          318         RLC                 ;ROTATE IT OVER ONE
01A6 DA9801      319         JC      LOOPK       ;DO IT AGAIN
01A9 3E00        320         MVI     A,00H       ;ZERO A
01AB 32EA0F      321         STA     KEYDWN      ;SAVE KEY DOWN
01AE C9          322         RET                 ;LEAVE
01AF 23          323 SAVKEY: INX     H           ;POINT AT RETURN LINE
01B0 2F          324         CMA                 ;PUT A BACK
01B1 77          325         MOV     M,A         ;SAVE RETURN LINE IN MEMORY
01B2 23          326         INX     H           ;POINT H AT SCAN LINE
01B3 70          327         MOV     M,B         ;SAVE SCAN LINE IN MEMORY
01B4 3E40        328         MVI     A,40H       ;SET A
01B6 32EA0F      329         STA     KEYDWN      ;SAVE KEY DOWN
01B9 C9          330         RET                 ;LEAVE
01BA 3E00        331 KYCHNG: MVI     A,00H       ;ZERO 0
01BC 32EA0F      332         STA     KEYDWN      ;RESET KEY DOWN
01BF C38F00      333         JMP     BYPASS      ;LEAVE
01C2 21F10F      334 KYDOWN: LXI     H,SCNLIN    ;GET SCAN LINE
01C5 7E          335         MOV     A,M         ;PUT SCAN LINE IN A
01C6 320018      336         STA     PORTA       ;OUTPUT SCAN LINE TO PORT A
01C9 2B          337         DCX     H           ;POINT AT RETURN LINE
01CA 3A0118      338         LDA     PORTB       ;GET RETURN LINES
01CD B6          339         ORA     M           ;ARE THEY THE SAME?
01CE 2F          340         CMA                 ;INVERT A
01CF B7          341         ORA     A           ;SET FLAGS
01D0 CABA01      342         JZ      KYCHNG      ;IF DIFFERENT KEY HAS CHANGED
01D3 3AEA0F      343         LDA     KEYDWN      ;GET KEY DOWN
01D6 E601        344         ANI     01H         ;HAS THIS BEEN DONE BEFORE?
01D8 C28F00      345         JNZ     BYPASS      ;LEAVE IF IT HAS
01DB 3A0118      346         LDA     PORTB       ;GET RETURN LINE
01DE 06FF        347         MVI     B,0FFH      ;GET READY TO ZERO B
01E0 04          348 UP:     INR     B           ;ZERO B
01E1 0F          349         RRC                 ;ROTATE A
01E2 DAE001      350         JC      UP          ;DO IT AGAIN
01E5 23          351         INX     H           ;POINT H AT SCAN LINES
01E6 7E          352         MOV     A,M         ;GET SCAN LINES
01E7 0EFF        353         MVI     C,0FFH      ;GET READY TO LOOP
01E9 0C          354 UP1:    INR     C           ;START C COUNTING
01EA 0F          355         RRC                 ;ROTATE A
01EB DAE901      356         JC      UP1         ;JUMP TO LOOP
01EE 78          357         MOV     A,B         ;GET RETURN LINES
01EF 07          358         RLC                 ;MOVE OVER ONCE
01F0 07          359         RLC                 ;MOVE OVER TWICE
01F1 07          360         RLC                 ;MOVE OVER THREE TIMES
01F2 B1          361         ORA     C           ;OR SCAN AND RETURN LINES
01F3 47          362         MOV     B,A         ;SAVE A IN B
01F4 3A0218      363         LDA     PORTC       ;GET SHIFT CONTROL
01F7 E640        364         ANI     40H         ;IS CONTROL SET
01F9 4F          365         MOV     C,A         ;SAVE A IN C
01FA 3AEF0F      366         LDA     SHCON       ;GET SHIFT CONTROL
01FD 57          367         MOV     D,A         ;SAVE A IN D
01FE E640        368         ANI     40H         ;STRIP CONTROL
0200 B1          369         ORA     C           ;SET BIT
0201 CA3E02      370         JZ      CNTDWN      ;IF SET LEAVE
0204 3A0218      371         LDA     PORTC       ;READ IT AGAIN
0207 E620        372         ANI     20H         ;STRIP SHIFT
0209 4F          373         MOV     C,A         ;SAVE A
020A 7A          374         MOV     A,D         ;GET SHIFT CONTROL
020B E620        375         ANI     20H         ;STRIP CONTROL
020D B1          376         ORA     C           ;ARE THEY THE SAME?
020E CA4702      377         JZ      SHDWN       ;IF SET LEAVE
0211 58          378 SCR:    MOV     E,B         ;PUT  TARGET IN E
0212 1600        379         MVI     D,00H       ;ZERO D
0214 210705      380         LXI     H,KYLKUP    ;GET LOOKUP TABLE
0217 19          381         DAD     D           ;GET OFFSET
0218 7E          382         MOV     A,M         ;GET CHARACTER
0219 47          383         MOV     B,A         ;PUT CHARACTER IN B
021A 3A0218      384         LDA     PORTC       ;GET PORTC
021D E610        385         ANI     10H         ;STRIP BIT
021F CA2E02      386         JZ      CAPLOC      ;CAPS LOCK
0222 78          387         MOV     A,B         ;GET A BACK
0223 32EB0F      388 STKEY:  STA     KBCHR       ;SAVE CHARACTER
0226 3EC1        389         MVI     A,0C1H      ;SET A
0228 32EA0F      390         STA     KEYDWN      ;SAVE KEY DOWN
022B C38F00      391         JMP     BYPASS      ;LEAVE
                 392         ;
                 393         ;IF THE CAP LOCK BUTTON IS PUSHED THIS ROUTINE SEES IF
                 394         ;THE CHARACTER IS BETWEEN 61H AND 7AH AND IF IT IS THIS
```

207780-39

```
                        ;ROUTINE ASSUMES THAT THE CHARACTER IS LOWER CASE ASCII
                        ;AND SUBTRACTS 20H, WHICH CONVERTS THE CHARACTER TO
                        ;UPPER CASE ASCII
                        ;
022E 78                 CAPLOC: MOV     A,B             ;GET A BACK
022F FE60               CPI     60H             ;HOW BIG IS IT?
0231 DA2302             JC      STKEY           ;LEAVE IF IT'S TOO SMALL
0234 FE7B               CPI     7BH             ;IS IT TOO BIG
0236 D22302             JNC     STKEY           ;LEAVE IF TOO BIG
0239 D620               SUI     20H             ;ADJUST A
023B C32302             JMP     STKEY           ;STORE THE KEY
                        ;
                        ;THE ROUTINES SHDWN AND CNTDWN SET BIT 6 AND 7 RESPECTIVLY
                        ;IN THE ACC.
                        ;
023E 3E80               CNTDWN: MVI     A,80H           ;SET BIT 7 IN A
0240 B0                 ORA     B               ;OR WITH CHARACTER
0241 E6BF               ANI     0BFH            ;MAKE SURE SHIFT IS NOT SET
0243 47                 MOV     B,A             ;PUT IT BACK IN B
0244 C31102             JMP     SCR             ;GO BACK
0247 3E40               SHDWN:  MVI     A,40H           ;SET BIT 6 IN A
0249 B0                 ORA     B               ;OR WITH CHARACTER
024A 47                 MOV     B,A             ;PUT IT BACK IN B
024B C31102             JMP     SCR             ;GO BACK
                        ;
                        ;THIS ROUTINE CHECKS FOR ESCAPE CHARACTERS, LF, CR,
                        ;FF, AND BACK SPACE
                        ;
024E 3AEE0F             CHREC:  LDA     ESCP            ;ESCAPE SET?
0251 FE80               CPI     80H             ;SEE IF IT IS
0253 CA7B02             JZ      ESSQ            ;LEAVE IF IT IS
0256 3AE70F             LDA     USCHR           ;GET CHARACTER
0259 FE0A               CPI     0AH             ;LINE FEED
025B CAF603             JZ      LNFD            ;GO TO LINE FEED
025E FE0C               CPI     0CH             ;FORM FEED
0260 CACA03             JZ      FMFD            ;GO TO FORM FEED
0263 FE0D               CPI     0DH             ;CR
0265 CAAD03             JZ      CRRT            ;DO A CR
0268 FE08               CPI     08H             ;BACK SPACE
026A CA6E03             JZ      LEFT            ;DO A BACK SPACE
026D FE1B               CPI     1BH             ;ESCAPE
026F CAA503             JZ      ESKAP           ;DO AN ESCAPE
0272 B7                 ORA     A               ;CLEAR CARRY
0273 C6E0               ADI     0E0H            ;SEE IF CHARACTER IS PRINTABLE
0275 DA7704             JC      CHRPUT          ;IF PRINTABLE DO IT
0278 C30F01             JMP     SETUP           ;GO BACK AND READ USART AGAIN
                        ;
                        ;THIS ROUTINE RESETS THE ESCAPE LOCATION AND DECODES
                        ;THE CHARACTERS FOLLOWING AN ESCAPE. THE COMMANDS ARE
                        ;COMPATABLE WITH INTELS CREDIT TEXT EDITOR
                        ;
027B 3E00               ESSQ:   MVI     A,00H           ;ZERO A
027D 32EE0F             STA     ESCP            ;RESET  ESCP
0280 3AE70F             LDA     USCHR           ;GET CHARACTER
0283 FE42               CPI     42H             ;DOWN
0285 CAAE02             JZ      DOWN            ;MOVE CURSOR DOWN
0288 FE45               CPI     45H             ;CLEAR SCREEN CHARACTER
028A CACF02             JZ      CLEAR           ;CLEAR THE SCREEN
028D FE4A               CPI     4AH             ;CLEAR REST OF SCREEN
028F CAD502             JZ      CLRST           ;GO CLEAR THE REST OF THE SCREEN
0292 FE4B               CPI     4BH             ;CLEAR LINE CHARACTER
0294 CA2703             JZ      CLRLIN          ;GO CLEAR A LINE
0297 FE41               CPI     41H             ;CURSOR UP CHARACTER
0299 CA3303             JZ      UPCUR           ;MOVE CURSOR UP
029C FE43               CPI     43H             ;CURSOR RIGHT CHARACTER
029E CA4503             JZ      RIGHT           ;MOVE CURSOR TO THE RIGHT
02A1 FE44               CPI     44H             ;CURSOR LEFT CHARACTER
02A3 CA6E03             JZ      LEFT            ;MOVE CURSOR TO THE LEFT
02A6 FE48               CPI     48H             ;HOME CURSOR CHARACTER
02A8 CA9703             JZ      HOME            ;HOME THE CURSOR
02AB C30F01             JMP     SETUP           ;LEAVE
                        ;
                        ;THIS ROUTINE MOVES THE CURSOR DOWN ONE CHARACTER LINE
                        ;
02AE 3AE10F             DOWN:   LDA     CURSY           ;PUT CURSOR Y IN A
02B1 FE18               CPI     CURBOT          ;SEE IF ON BOTTOM OF SCREEN
02B3 CA0F01             JZ      SETUP           ;LEAVE IF ON BOTTOM
02B6 3C                 INR     A               ;INCREMENT Y CURSOR
02B7 32E10F             STA     CURSY           ;SAVE NEW CURSOR
02BA CDB803             CALL    LDCUR           ;LOAD THE CURSOR
02BD CDA504             CALL    CALCU           ;CALCULATE ADDRESS
02C0 7E                 MOV     A,M             ;GET FIRST LOCATION OF THE LINE
02C1 FEF0               CPI     0F0H            ;SEE IF CLEAR SCREEN CHARACTER
02C3 C20F01             JNZ     SETUP           ;LEAVE IF IT IS NOT
02C6 22E50F             SHLD    LOC80           ;SAVE BEGINNING OF THE LINE
02C9 CD1504             CALL    CLLINE          ;CLEAR THE LINE
02CC C30F01             JMP     SETUP           ;LEAVE
                        ;
```

207780-40

```
                        ;THIS ROUTINE CLEARS THE SCREEN.
                        ;
02CF CDE403     CLEAR:  CALL    CLSCR           ;GO CLEAR THE SCREEN
02D2 C30F01             JMP     SETUP           ;GO BACK
                        ;
                        ;THIS ROUTINE CLEARS ALL LINES BENEATH THE LOCATION
                        ;OF THE CURSOR.
                        ;
02D5 CDA504     CLRST:  CALL    CALCU           ;CALCULATE ADDRESS
02D8 CDCD04             CALL    ADX             ;ADD X POSITION
02DB 01204F             LXI     B,4F20H         ;PUT SPACE AND LAST X IN B AND C
02DE 3AE20F             LDA     CURSX           ;GET X CURSOR
02E1 B8                 CMP     B               ;SEE IF AT END OF LINE
02E2 CABC02             JZ      OVR1            ;LEAVE IF X IS AT END OF LINE
02E5 3C         LLP:    INR     A               ;MOVE A OVER ONE X POSITION
02E6 23                 INX     H               ;INCREMENT MEMORY POINTER
02E7 71                 MOV     M,C             ;PUT A SPACE IN MEMORY
02E8 B8                 CMP     B               ;SEE IF A = 4FH
02E9 C2E502             JNZ     LLP             ;IF NOT LOOP AGAIN
02EC 01D00F     OVR1:   LXI     B,LAST          ;PUT LAST LINE IN BC
02EF 23                 INX     H               ;POINT HL TO LAST LINE
02F0 78                 MOV     A,B             ;GET B
02F1 BC                 CMP     H               ;SAME AS H?
02F2 C2FD02             JNZ     CONCL           ;LEAVE IF NOT
02F5 79                 MOV     A,C             ;GET C
02F6 BD                 CMP     L               ;SAME AS L?
02F7 C2FD02             JNZ     CONCL           ;LEAVE IF NOT
02FA 210008             LXI     H,TPDIS         ;GET TOP OF DISPLAY
02FD 3AE10F     CONCL:  LDA     CURSY           ;GET Y CURSOR
0300 FE18               CPI     CURBOT          ;IS IT ON THE BOTTOM
0302 CA0F01             JZ      SETUP           ;LEAVE IF IT IS
0305 3C                 INR     A               ;MOVE IT DOWN ONE LINE
0306 47                 MOV     B,A             ;SAVE CURSOR IN B FOR LATER
0307 115000             LXI     D,LNGTH         ;PUT LENGTH OF ONE LINE IN D
030A 36F0       CLOOP:  MVI     M,0F0H          ;PUT EOR IN MEMORY
030C 78                 MOV     A,B             ;GET CURSOR Y
030D FE18               CPI     CURBOT          ;ARE WE ON THE BOTTOM
030F CA0F01             JZ      SETUP           ;LEAVE IF WE ARE
0312 3C                 INR     A               ;MOVE CURSOR DOWN ONE
0313 19                 DAD     D               ;GET NEXT LINE
0314 47                 MOV     B,A             ;SAVE A
0315 7C                 MOV     A,H             ;PUT H IN A
0316 FE0F               CPI     0FH             ;COMPARE TO HIGH LAST
0318 C20A03             JNZ     CLOOP           ;LEAVE IF IT IS NOT
0318 7D                 MOV     A,L             ;PUT L IN A
031C FED0               CPI     0D0H            ;COMPARE TO LOW LAST
031E C20A03             JNZ     CLOOP           ;LEAVE IF IT IS NOT
0321 210008             LXI     H,TPDIS         ;PUT TOP DISPLAY IN H AND L
0324 C30A03             JMP     CLOOP           ;LOOP AGAIN
                        ;
                        ;THIS ROUTINE CLEARS THE LINE THE CURSOR IS ON.
                        ;
0327 CDA504     CLRLIN: CALL    CALCU           ;CALCULATE ADDRESS
032A 22E50F             SHLD    LOC80           ;STORE H AND L TO CLEAR LINE
032D CD1504             CALL    CLLINE          ;CLEAR THE LINE
0330 C30F01             JMP     SETUP           ;GO BACK
                        ;
                        ;THIS ROUTINE MOVES THE CURSOR UP ONE LINE.
                        ;
0333 3AE10F     UPCUR:  LDA     CURSY           ;GET Y CURSOR
0336 FE00               CPI     00H             ;IS IT ZERO
0338 CA0F01             JZ      SETUP           ;IF IT IS LEAVE
033B 3D                 DCR     A               ;MOVE CURSOR UP
033C 32E10F             STA     CURSY           ;SAVE NEW CURSOR
033F CD8803             CALL    LDCUR           ;LOAD THE CURSOR
0342 C30F01             JMP     SETUP           ;LEAVE
                        ;
                        ;THIS ROUTINE MOVES THE CURSOR ONE LOCATION TO THE RIGHT
                        ;
0345 3AE20F     RIGHT:  LDA     CURSX           ;GET X CURSOR
0348 FE4F               CPI     4FH             ;IS IT ALL THE WAY OVER?
034A C25403             JNZ     NTOVER          ;IF NOT JUMP AROUND
034D 3AE10F             LDA     CURSY           ;GET Y CURSOR
0350 FE18               CPI     CURBOT          ;SEE IF ON BOTTOM
0352 CA5903             JZ      GD18            ;IF WE ARE JUMP
0355 3C                 INR     A               ;INCREMENT Y CURSOR
0356 32E10F             STA     CURSY           ;SAVE IT
0359 3E00       GD18:   MVI     A,00H           ;ZERO A
035B 32E20F             STA     CURSX           ;ZERO X CURSOR
035E CD8803             CALL    LDCUR           ;LOAD THE CURSOR
0361 C30F01             JMP     SETUP           ;LEAVE
0364 3C         NTOVER: INR     A               ;INCREMENT X CURSOR
0365 32E20F             STA     CURSX           ;SAVE IT
0368 CD8803             CALL    LDCUR           ;LOAD THE CURSOR
036B C30F01             JMP     SETUP           ;LEAVE
                        ;
                        ;THIS ROUTINE MOVES THE CURSOR LEFT ONE CHARACTER POSITION
```

207780-41

```
                        570 ;
036E 3AE20F             571 LEFT:   LDA     CURSX           ;GET X CURSOR
0371 FE00               572         CPI     00H             ;IS IT ALL THE WAY OVER
0373 C28D03             573         JNZ     NOVER           ;IF NOT JUMP AROUND
0376 3AE10F             574         LDA     CURSY           ;GET CURSOR Y
0379 FE00               575         CPI     00H             ;IS IT ZERO?
037B CA0F01             576         JZ      SETUP           ;IF IT IS JUMP
037E 3D                 577         DCR     A               ;MOVE CURSOR Y UP
037F 32E10F             578         STA     CURSY           ;SAVE IT
0382 3E4F               579         MVI     A,4FH           ;GET LAST X LOCATION
0384 32E20F             580         STA     CURSX           ;SAVE IT
0387 CDB803             581         CALL    LDCUR           ;LOAD THE CURSOR
038A C30F01             582         JMP     SETUP
038D 3D                 583 NOVER:  DCR     A               ;ADJUST X CURSOR
038E 32E20F             584         STA     CURSX           ;SAVE CURSOR X
0391 CDB803             585         CALL    LDCUR           ;LOAD THE CURSOR
0394 C30F01             586         JMP     SETUP           ;LEAVE
                        587 ;
                        588 ;THIS ROUTINE HOMES THE CURSOR.
                        589 ;
0397 3E00               590 HOME:   MVI     A,00H           ;ZERO A
0399 32E20F             591         STA     CURSX           ;ZERO X CURSOR
039C 32E10F             592         STA     CURSY           ;ZERO Y CURSOR
039F CDB803             593         CALL    LDCUR           ;LOAD THE CURSOR
03A2 C30F01             594         JMP     SETUP           ;LEAVE
                        595 ;
                        596 ;THIS ROUTINE SETS THE ESCAPE BIT
                        597 ;
03A5 3E80               598 ESKAP:  MVI     A,80H           ;LOAD A WITH ESCAPE BIT
03A7 32EE0F             599         STA     ESCP            ;SET ESCAPE LOCATION
03AA C30F01             600         JMP     SETUP           ;GO BACK AND READ USART
                        601 ;
                        602 ;THIS ROUTINE DOES A CR
                        603 ;
03AD 3E00               604 CGRT:   MVI     A,00H           ;ZERO A
03AF 32E20F             605         STA     CURSX           ;ZERO  CURSOR X
03B2 CDB803             606         CALL    LDCUR           ;LOAD CURSOR INTO 8275
03B5 C30F01             607         JMP     SETUP           ;POLL USART AGAIN
                        608 ;
                        609 ;THIS ROUTINE LOADS THE CURSOR
                        610 ;
03B8 3E80               611 LDCUR:  MVI     A,80H           ;PUT 80H INTO A
03BA 320110             612         STA     CRTS            ;LOAD CURSOR INTO 8275
03BD 3AE20F             613         LDA     CURSX           ;GET CURSOR X
03C0 320010             614         STA     CRTM            ;PUT IT IN 8275
03C3 3AE10F             615         LDA     CURSY           ;GET CURSOR Y
03C6 320010             616         STA     CRTM            ;PUT IT IN 8275
03C9 C9                 617         RET
                        618 ;
                        619 ;THIS ROUTINE DOES A FORM FEED
                        620 ;
03CA CDE403             621 FMFD:   CALL    CLSCR           ;CALL CLEAR SCREEN
03CD 210008             622         LXI     H,TPDIS         ;PUT TOP DISPLAY IN HL
03D0 22E50F             623         SHLD    LOC80           ;PUT IT IN LOC80
03D3 CD1504             624         CALL    CLLINE          ;CLEAR TOP LINE
03D6 3E00               625         MVI     A,00H           ;ZERO A
03D8 32E20F             626         STA     CURSX           ;ZERO CURSOR X
03DB 32E10F             627         STA     CURSY           ;ZERO CURSOR Y
03DE CDB803             628         CALL    LDCUR           ;LOAD THE CURSOR
03E1 C30F01             629         JMP     SETUP           ;BACK TO USART
                        630 ;
                        631 ;THIS ROUTINE CLEARS THE SCREEN BY WRITING END OF ROW
                        632 ;CHARACTERS INTO THE FIRST LOCATION OF ALL LINES ON
                        633 ;THE SCREEN.
                        634 ;
03E4 3EF0               635 CLSCR:  MVI     A,0F0H          ;PUT EOR CHARACTER IN A
03E6 0618               636         MVI     B,CURBOT        ;LOAD B WITH MAX Y
03E8 04                 637         INR     B               ;GO TO MAX PLUS ONE
03E9 210008             638         LXI     H,TPDIS         ;LOAD H AND L WITH TOP OF RAM
03EC 115000             639         LXI     D,LNGTH         ;MOVE 50H = 80D INTO D AND E
03EF 77                 640 LOADX:  MOV     M,A             ;MOVE EOR INTO MEMORY
03F0 19                 641         DAD     D               ;CHANGE POINTER BY 80D
03F1 05                 642         DCR     B               ;COUNT THE LOOPS
03F2 C2EF03             643         JNZ     LOADX           ;CONTINUE IF NOT ZERO
03F5 C9                 644         RET                     ;GO BACK
                        645 ;
                        646 ;THIS ROUTINE DOES A LINE FEED
                        647 ;
03F6 CDFC03             648 LNFD:   CALL    LNFD1           ;CALL ROUTINE
03F9 C30F01             649         JMP     SETUP           ;POLL FLAGS
                        650 ;
                        651 ;LINE FEED
                        652 ;
03FC 3AE10F             653 LNFD1:  LDA     CURSY           ;GET Y LOCATION OF CURSOR
03FF FE18               654         CPI     CURBOT          ;SEE IF AT BOTTOM OF SCREEN
0401 CA5304             655         JZ      ONBOT           ;IF WE ARE, LEAVE
0404 3C                 656         INR     A               ;INCREMENT A
0405 32E10F             657         STA     CURSY           ;SAVE NEW CURSOR
```

207780-42

```
0408 CDA504          658          CALL    CALCU           ;CALCULATE ADDRESS
0408 22E50F          659          SHLD    LOC80           ;SAVE TO CLEAR LINE
040E CD1504          660          CALL    CLLINE          ;CLEAR THE LINE
0411 CD8803          661          CALL    LDCUR           ;LOAD THE CURSOR
0414 C9              662          RET                     ;LEAVE
                     663          ;
                     664          ;THIS ROUTINE CLEARS THE LINE WHOSE FIRST ADDRESS
                     665          ;IS IN LOC80. PUSH INSTRUCTIONS ARE USED TO RAPIDLY
                     666          ;CLEAR THE LINE
                     667          ;
0415 F3              668 CLLINE:  DI                      ;NO INTERRUPTS HERE
0416 2AE50F          669          LHLD    LOC80           ;GET LOC80
0419 115000          670          LXI     D,LNGTH         ;GET OFFSET
041C 19              671          DAD     D               ;ADD OFFSET
041D EB              672          XCHG                    ;PUT START IN DE
041E 210000          673          LXI     H,0000H         ;ZERO HL
0421 39              674          DAD     SP              ;GET STACK
0422 EB              675          XCHG                    ;PUT STACK IN DE
0423 F9              676          SPHL                    ;PUT START IN SP
0424 212020          677          LXI     H,2020H         ;PUT SPACES IN HL
                     678          ;
                     679          ;NOW DO 40 PUSH INSTRUCTIONS TO CLEAR THE LINE
                     680          ;
                     681          REPT (LNGTH/2)
                     682          PUSH    H
                     683          ENDM
0427 E5              684+         PUSH    H
0428 E5              685+         PUSH    H
0429 E5              686+         PUSH    H
042A E5              687+         PUSH    H
042B E5              688+         PUSH    H
042C E5              689+         PUSH    H
042D E5              690+         PUSH    H
042E E5              691+         PUSH    H
042F E5              692+         PUSH    H
0430 E5              693+         PUSH    H
0431 E5              694+         PUSH    H
0432 E5              695+         PUSH    H
0433 E5              696+         PUSH    H
0434 E5              697+         PUSH    H
0435 E5              698+         PUSH    H
0436 E5              699+         PUSH    H
0437 E5              700+         PUSH    H
0438 E5              701+         PUSH    H
0439 E5              702+         PUSH    H
043A E5              703+         PUSH    H
043B E5              704+         PUSH    H
043C E5              705+         PUSH    H
043D E5              706+         PUSH    H
043E E5              707+         PUSH    H
043F E5              708+         PUSH    H
0440 E5              709+         PUSH    H
0441 E5              710+         PUSH    H
0442 E5              711+         PUSH    H
0443 E5              712+         PUSH    H
0444 E5              713+         PUSH    H
0445 E5              714+         PUSH    H
0446 E5              715+         PUSH    H
0447 E5              716+         PUSH    H
0448 E5              717+         PUSH    H
0449 E5              718+         PUSH    H
044A E5              719+         PUSH    H
044B E5              720+         PUSH    H
044C E5              721+         PUSH    H
044D E5              722+         PUSH    H
044E E5              723+         PUSH    H
044F EB              724          XCHG                    ;PUT STACK IN HL
0450 F9              725          SPHL                    ;PUT IT BACK IN SP
0451 FB              726          EI                      ;ENABLE INTERRUPTS
0452 C9              727          RET                     ;GO BACK
                     728          ;
                     729          ;IF CURSOR IS ON THE BOTTOM OF THE SCREEN THIS ROUTINE
                     730          ;IS USED TO IMPLEMENT THE LINE FEED
                     731          ;
0453 2AE30F          732 ONBOT:   LHLD    TOPAD           ;GET TOP ADDRESS
0456 22E50F          733          SHLD    LOC80           ;SAVE IT IN LOC80
0459 115000          734          LXI     D,LNGTH         ;LINE LENGTH
045C 19              735          DAD     D               ;ADD HL + DE
045D 01D00F          736          LXI     B,LAST          ;GET BOTTOM LINE
0460 7C              737          MOV     A,H             ;GET H
0461 B8              738          CMP     B               ;SAME AS B
0462 C26D04          739          JNZ     ARND            ;LEAVE IF NOT SAME
0465 7D              740          MOV     A,L             ;GET L
0466 B9              741          CMP     C               ;SAME AS C
0467 C26D04          742          JNZ     ARND            ;LEAVE IF NOT SAME
046A 210000          743          LXI     H,TPDIS         ;LOAD HL WITH TOP OF DISPLAY
046D 22E30F          744 ARND:    SHLD    TOPAD           ;SAVE NEW TOP ADDRESS
```

207780-43

```
0470 CD1504      745          CALL    CLLINE          ;CLEAR LINE
0473 CD8803      746          CALL    LDCUR           ;LOAD THE CURSOR
0476 C9          747          RET
                 748          ;
                 749          ;THIS ROUTINE PUTS A CHARACTER ON THE SCREEN AND
                 750          ;INCREMENTS THE X CURSOR POSITION. A LINE FEED IS
                 751          ;INSERTED IF THE INCREMENTED CURSOR EQUALS 81D
                 752          ;
0477 CDA504      753 CHRPUT:  CALL    CALCU           ;CALCULATE SCREEN POSITION
047A 7E          754          MOV     A,M             ;GET FIRST CHARACTER
047B FEF0        755          CPI     0F0H            ;IS IT A CLEAR LINE
047D 22E50F      756          SHLD    LOC80           ;SAVE LINE TO CLEAR
0480 CC1504      757          CZ      CLLINE          ;CLEAR LINE
0483 2AE50F      758          LHLD    LOC80           ;GET LINE
0486 CDCD04      759          CALL    ADX             ;ADD CURSOR X
0489 3AE70F      760          LDA     USCHR           ;GET CHARACTER
048C 77          761          MOV     M,A             ;PUT IT ON SCREEN
048D 3AE20F      762          LDA     CURSX           ;GET CURSOR X
0490 3C          763          INR     A               ;INCREMENT CURSOR X
0491 FE50        764          CPI     LNGTH           ;HAS IT GONE TOO FAR?
0493 C29C04      765          JNZ     OK1             ;IF NOT GOOD
0496 CDFC03      766          CALL    LNFD1           ;DO A LINE FEED
0499 C3AD03      767          JMP     CGRT            ;DO A CR
049C 32E20F      768 OK1:     STA     CURSX           ;SAVE CURSOR
049F CD8803      769          CALL    LDCUR           ;LOAD THE CURSOR
04A2 C30F01      770          JMP     SETUP           ;LEAVE
                 771          ;
                 772          ;THIS ROUTINE TAKES THE TOP ADDRESS AND THE Y CURSOR
                 773          ;LOCATION AND CALCULATES THE ADDRESS OF THE LINE
                 774          ;THAT THE CURSOR IS ON. THE RESULT IS RETURNED IN H
                 775          ;AND L AND ALL REGISTERS ARE USED.
                 776          ;
04A5 21D504      777 CALCU:   LXI     H,LINTAB        ;GET LINE TABLE INTO H AND L
04A8 3AE10F      778          LDA     CURSY           ;GET CURSOR INTO A
04AB 07          779          RLC                     ;SET UP A FOR LOOKUP TABLE
04AC 0600        780          MVI     B,00H           ;ZERO B
04AE 4F          781          MOV     C,A             ;PUT CURSOR INTO A
04AF 09          782          DAD     B               ;ADD LINE TABLE TO Y CURSOR
04B0 7E          783          MOV     A,M             ;PUT LOW LINE TABLE INTO A
04B1 4F          784          MOV     C,A             ;PUT LOW LINE TABLE INTO C
04B2 23          785          INX     H               ;CHANGE MEMORY POINTER
04B3 7E          786          MOV     A,M             ;PUT HIGH LINE TABLE INTO A
04B4 47          787          MOV     B,A             ;PUT HIGH LINE TABLE INTO B
04B5 2100F8      788          LXI     H,0F800H        ;TWOS COMPLEMENT SCREEN LOCATION
04B8 09          789          DAD     B               ;SUBTRACT OFFSET
04B9 EB          790          XCHG                    ;SAVE HL IN DE
04BA 2AE30F      791          LHLD    TOPAD           ;GET TOP ADDRESS IN H AND L
04BD 19          792          DAD     D               ;GET DISPLACED ADDRESS
04BE D5          793          XCHG                    ;SAVE IT IN D
04BF 2130F0      794          LXI     H,0F030H        ;TWOS COMPLEMENT SCREEN LOCATION
04C2 19          795          DAD     D               ;SEE IF WE ARE OFF THE SCREEN
04C3 DAC804      796          JC      FIX             ;IF WE ARE FIX IT
04C6 EB          797          XCHG                    ;GET DISPLACED ADDRESS BACK
04C7 C9          798          RET                     ;GO BACK
04C8 2130F8      799 FIX:     LXI     H,0F830H        ;SCREEN BOUNDRY
04CB 19          800          DAD     D               ;ADJUST SCREEN
04CC C9          801          RET                     ;GO BACK
                 802          ;
                 803          ;THIS ROUTINE ADDS THE X CURSOR LOCATION TO THE ADDRESS
                 804          ;THAT IS IN THE H AND L REGISTERS AND RETURNS THE RESULT
                 805          ;IN H AND L
                 806          ;
04CD 3AE20F      807 ADX:     LDA     CURSX           ;GET CURSOR
04D0 0600        808          MVI     B,00H           ;ZERO B
04D2 4F          809          MOV     C,A             ;PUT CURSOR X IN C
04D3 09          810          DAD     B               ;ADD CURSOR X TO H AND L
04D4 C9          811          RET                     ;LEAVE
                 812          ;
                 813          ;THIS TABLE CONTAINS THE OFFSET ADDRESSES FOR EACH
                 814          ;OF THE 25 DISPLAYED LINES.
                 815          ;
0000             816 LINTAB:  LINNUM SET 0
                 817          REPT (CURBOT+1)
                 818          DW      TPDIS+(LNGTH*LINNUM)
                 819          LINNUM SET (LINNUM+1)
                 820          ENDM
04D5 0008        821+         DW      TPDIS+(LNGTH*LINNUM)
0001             822+         LINNUM SET (LINNUM+1)
04D7 5008        823+         DW      TPDIS+(LNGTH*LINNUM)
0002             824+         LINNUM SET (LINNUM+1)
04D9 A008        825+         DW      TPDIS+(LNGTH*LINNUM)
0003             826+         LINNUM SET (LINNUM+1)
04DB F008        827+         DW      TPDIS+(LNGTH*LINNUM)
0004             828+         LINNUM SET (LINNUM+1)
04DD 4009        829+         DW      TPDIS+(LNGTH*LINNUM)
0005             830+         LINNUM SET (LINNUM+1)
04DF 9009        831+         DW      TPDIS+(LNGTH*LINNUM)
```

207780-44

```
0006                    832+        LINNUM SET (LINNUM+1)
04E1 E009               833+        DW      TPDIS+(LNGTH*LINNUM)
0007                    834+        LINNUM SET (LINNUM+1)
04E3 300A               835+        DW      TPDIS+(LNGTH*LINNUM)
0008                    836+        LINNUM SET (LINNUM+1)
04E5 800A               837+        DW      TPDIS+(LNGTH*LINNUM)
0009                    838+        LINNUM SET (LINNUM+1)
04E7 D00A               839+        DW      TPDIS+(LNGTH*LINNUM)
000A                    840+        LINNUM SET (LINNUM+1)
04E9 200B               841+        DW      TPDIS+(LNGTH*LINNUM)
000B                    842+        LINNUM SET (LINNUM+1)
04EB 700B               843+        DW      TPDIS+(LNGTH*LINNUM)
000C                    844+        LINNUM SET (LINNUM+1)
04ED C00B               845+        DW      TPDIS+(LNGTH*LINNUM)
000D                    846+        LINNUM SET (LINNUM+1)
04EF 100C               847+        DW      TPDIS+(LNGTH*LINNUM)
000E                    848+        LINNUM SET (LINNUM+1)
04F1 600C               849+        DW      TPDIS+(LNGTH*LINNUM)
000F                    850+        LINNUM SET (LINNUM+1)
04F3 B00C               851+        DW      TPDIS+(LNGTH*LINNUM)
0010                    852+        LINNUM SET (LINNUM+1)
04F5 000D               853+        DW      TPDIS+(LNGTH*LINNUM)
0011                    854+        LINNUM SET (LINNUM+1)
04F7 500D               855+        DW      TPDIS+(LNGTH*LINNUM)
0012                    856+        LINNUM SET (LINNUM+1)
04F9 A00D               857+        DW      TPDIS+(LNGTH*LINNUM)
0013                    858+        LINNUM SET (LINNUM+1)
04FB F00D               859+        DW      TPDIS+(LNGTH*LINNUM)
0014                    860+        LINNUM SET (LINNUM+1)
04FD 400E               861+        DW      TPDIS+(LNGTH*LINNUM)
0015                    862+        LINNUM SET (LINNUM+1)
04FF 900E               863+        DW      TPDIS+(LNGTH*LINNUM)
0016                    864+        LINNUM SET (LINNUM+1)
0501 E00E               865+        DW      TPDIS+(LNGTH*LINNUM)
0017                    866+        LINNUM SET (LINNUM+1)
0503 300F               867+        DW      TPDIS+(LNGTH*LINNUM)
0018                    868+        LINNUM SET (LINNUM+1)
0505 800F               869+        DW      TPDIS+(LNGTH*LINNUM)
0019                    870+        LINNUM SET (LINNUM+1)
                        871         ;
                        872         ;KEYBOARD LOOKUP TABLE
                        873         ;THIS TABLE CONTAINS ALL THE ASCII CHARACTERS
                        874         ;THAT ARE TRANSMITTED BY THE TERMINAL
                        875         ;THE CHARACTERS ARE ORGANIZED SO THAT BITS 0,1 AND 2
                        876         ;ARE THE SCAN LINES, BITS 3,4 AND 5 ARE THE RETURN LINES
                        877         ;BIT 6 IS SHIFT AND BIT 7 IS CONTROL
                        878         ;
0507 38                 879 KYLKUP: DB      38H,39H         ;8 AND 9
0508 39
0509 30                 880         DB      30H,2DH         ;0 AND -
050A 2D
050B 3D                 881         DB      3DH,5CH         ;= AND \
050C 5C
050D 08                 882         DB      08H,00H         ;BS AND BREAK
050E 00
050F 75                 883         DB      75H,69H         ;LOWER CASE U AND I
0510 69
0511 6F                 884         DB      6FH,70H         ;LOWER CASE O AND P
0512 70
0513 5B                 885         DB      5BH,5CH         ;[ AND \
0514 5C
0515 0A                 886         DB      0AH,7FH         ;LF AND DELETE
0516 7F
0517 6A                 887         DB      6AH,6BH         ;LOWER CASE J AND K
0518 6B
0519 6C                 888         DB      6CH,3BH         ;LOWER CASE L AND ;
051A 3B
051B 27                 889         DB      27H,00H         ;' AND NOTHING
051C 00
051D 0D                 890         DB      0DH,37H         ;CR AND 7
051E 37
051F 6D                 891         DB      6DH,2CH         ;LOWER CASE M AND COMMA
0520 2C
0521 2E                 892         DB      2EH,2FH         ;PERIOD AND SLASH
0522 2F
0523 00                 893         DB      00H,00H         ;BLANK AND NOTHING
0524 00
0525 00                 894         DB      00H,00H         ;NOTHING AND NOTHING
0526 00
0527 00                 895         DB      00H,61H         ;NOTHING AND LOWER CASE A
0528 61
0529 7A                 896         DB      7AH,78H         ;LOWER CASE Z AND X
052A 78
052B 63                 897         DB      63H,76H         ;LOWER CASE C AND V
052C 76
052D 62                 898         DB      62H,6EH         ;LOWER CASE B AND N
052E 6E
```

207780-45

```
052F 79          899     DB      79H,00H         ;LOWER CASE Y AND NOTHING
0530 00
0531 00          900     DB      00H,20H         ;NOTHING AND SPACE
0532 20
0533 64          901     DB      64H,66H         ;LOWER CASE D AND F
0534 66
0535 67          902     DB      67H,68H         ;LOWER CASE G AND H
0536 68
0537 00          903     DB      00H,71H         ;TAB AND LOWER CASE Q
0538 71
0539 77          904     DB      77H,73H         ;LOWER CASE W AND S
053A 73
053B 65          905     DB      65H,72H         ;LOWER CASE E AND R
053C 72
053D 74          906     DB      74H,00H         ;LOWER CASE T AND NOTHING
053E 00
053F 1B          907     DB      1BH,31H         ;ESCAPE AND 1
0540 31
0541 32          908     DB      32H,33H         ; 2 AND 3
0542 33
0543 34          909     DB      34H,35H         ; 4 AND 5
0544 35
0545 36          910     DB      36H,00H         ; 6 AND NOTHING
0546 00
0547 2A          911     DB      2AH,28H         ;* AND )
0548 28
0549 29          912     DB      29H,5FH         ;( AND -
054A 5F
054B 2B          913     DB      2BH,00H         ;+ AND NOTHING
054C 00
054D 08          914     DB      08H,00H         ;BS AND BREAK
054E 00
054F 55          915     DB      55H,49H         ;U AND I
0550 49
0551 4F          916     DB      4FH,50H         ;O AND P
0552 50
0553 5D          917     DB      5DH,00H         ;] AND NO CHARACTER
0554 00
0555 0A          918     DB      0AH,7FH         ;LF AND DELETE
0556 7F
0557 4A          919     DB      4AH,4BH         ;J AND K
0558 4B
0559 4C          920     DB      4CH,3AH         ;L AND :
055A 3A
055B 22          921     DB      22H,00H         ;" AND NO CHARACTER
055C 00
055D 0D          922     DB      0DH,26H         ;CR AND &
055E 26
055F 4D          923     DB      4DH,3CH         ;M AND <
0560 3C
0561 3E          924     DB      3EH,3FH         ;> AND ?
0562 3F
0563 00          925     DB      00H,00H         ;BLANK AND NOTHING
0564 00
0565 00          926     DB      00H,00H         ;NOTHING AND NOTHING
0566 00
0567 00          927     DB      00H,41H         ;NOTHING AND A
0568 41
0569 5A          928     DB      5AH,58H         ;Z AND X
056A 58
056B 43          929     DB      43H,56H         ;C AND V
056C 56
056D 42          930     DB      42H,4EH         ;B AND N
056E 4E
056F 59          931     DB      59H,00H         ;Y AND NOTHING
0570 00
0571 00          932     DB      00H,20H         ;NO CHARACTER AND SPACE
0572 20
0573 44          933     DB      44H,46H         ;D AND F
0574 46
0575 47          934     DB      47H,48H         ;G AND H
0576 48
0577 00          935     DB      00H,51H         ;TAB AND Q
0578 51
0579 57          936     DB      57H,53H         ;W AND S
057A 53
057B 45          937     DB      45H,52H         ;E AND R
057C 52
057D 54          938     DB      54H,00H         ;T AND NO CONNECTION
057E 00
057F 1B          939     DB      1BH,21H         ;ESCAPE AND !
0580 21
0581 40          940     DB      40H,23H         ;@ AND #
0582 23
0583 24          941     DB      24H,25H         ;$ AND %
0584 25
0585 5E          942     DB      5EH,00H         ;^ AND NO CONNECTION
```

207780-46

```
0586 00
                        943     ;
                        944     ;THIS IS WHERE THE CONTROL CHARACTERS ARE LOOKED UP
                        945     ;
0587 00                 946     DB      00H,00H         ;NOTHING
0588 00
0589 00                 947     DB      00H,00H         ;NOTHING
058A 00
058B 00                 948     DB      00H,00H         ;NOTHING
058C 00
058D 00                 949     DB      00H,00H         ;NOTHING
058E 00
058F 15                 950     DB      15H,09H         ;CONTROL U AND I
0590 09
0591 0F                 951     DB      0FH,10H         ;CONTROL O AND P
0592 10
0593 0B                 952     DB      0BH,0CH         ;CONTROL [ AND \
0594 0C
0595 0A                 953     DB      0AH,7FH         ;LF AND DELETE
0596 7F
0597 0A                 954     DB      0AH,08H         ;CONTROL J AND K
0598 0B
0599 0C                 955     DB      0CH,00H         ;CONTROL L AND NOTHING
059A 00
059B 00                 956     DB      00H,00H         ;NOTHING
059C 00
059D 0D                 957     DB      0DH,00H         ;CR AND NOTHING
059E 00
059F 0D                 958     DB      0DH,00H         ;CONTROL M AND COMMA
05A0 00
05A1 00                 959     DB      00H,00H         ;NOTHING
05A2 00
05A3 00                 960     DB      00H,00H         ;NOTHING
05A4 00
05A5 00                 961     DB      00H,00H         ;NOTHING AND NOTHING
05A6 00
05A7 1A                 962     DB      1AH,18H         ;CONTROL Z AND X
05A8 18
05A9 03                 963     DB      03H,16H         ;CONTROL C AND V
05AA 16
05AB 02                 964     DB      02H,0EH         ;CONTROL B AND N
05AC 0E
05AD 19                 965     DB      19H,00H         ;CONTROL Y AND NOTHING
05AE 00
05AF 00                 966     DB      00H,20H         ;NOTHING AND SPACE
05B0 20
05B1 04                 967     DB      04H,06H         ;CONTROL D AND F
05B2 06
05B3 07                 968     DB      07H,08H         ;CONTROL G AND H
05B4 08
05B5 00                 969     DB      00H,11H         ;NOTHING AND CONTROL Q
05B6 11
05B7 17                 970     DB      17H,13H         ;CONTROL W AND S
05B8 13
05B9 05                 971     DB      05H,12H         ;CONTROL E AND R
05BA 12
05BB 14                 972     DB      14H,00H         ;CONTROL W AND NOTHING
05BC 00
05BD 1B                 973     DB      1BH,1DH         ;ESCAPE AND HOME(CREDIT)
05BE 1D
05BF 1E                 974     DB      1EH,1CH         ;CURSOR UP AND DOWN(CREDIT)
05C0 1C
05C1 14                 975     DB      14H,1FH         ;CURSOR RIGHT AND LEFT(CREDIT)
05C2 1F
05C3 00                 976     DB      00H,00H         ;NOTHING
05C4 00
                        977     ;
                        978     ;LOOK UP TABLE FOR 8253 BAUD RATE GENERATOR
                        979     ;
05C5 00                 980 BDLK:   DB      00H,05H,69H,03H ;75 AND 110 BAUD
05C6 05
05C7 69
05C8 03
05C9 80                 981     DB      80H,02H,40H,01H ;150 AND 300 BAUD
05CA 02
05CB 40
05CC 01
05CD A0                 982     DB      0A0H,00H        ;600 BAUD
05CE 00
05CF 50                 983     DB      50H,00H         ;1200 BAUD
05D0 00
05D1 28                 984     DB      28H,00H         ;2400 BAUD
05D2 00
05D3 14                 985     DB      14H,00H         ;4800 BAUD
05D4 00
05D5 0A                 986     DB      0AH,00H         ;9600 BAUD
05D6 00
```

207780-47

```
                        987             ;
                        988             ;DATA AREA
                        989             ;
0FE1                    990             ORG     0FE1H
0001                    991 CURSY:      DS      1
0001                    992 CURSX:      DS      1
0002                    993 TOPAD:      DS      2
0002                    994 LOC80:      DS      2
0001                    995 USCHR:      DS      1
0002                    996 CURAD:      DS      2
0001                    997 KEYDWN:     DS      1
0001                    998 KBCHR:      DS      1
0001                    999 BAUD:       DS      1
0001                   1000 KEYOK:      DS      1
0001                   1001 ESCP:       DS      1
0001                   1002 SHCON:      DS      1
0001                   1003 RETLIN:     DS      1
0001                   1004 SCNLIN:     DS      1
                       1005             END

PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
ADX      A 04CD   ARND     A 046D   BAUD     A 0FEC   BDLK     A 05C5   BTDIS    A 0F80   BYPASS   A 008F
CAPLOC   A 022E   CGRT     A 03AD   CHREC    A 024E   CHRPUT   A 0477   CLEAR    A 02CF   CLLINE   A 0415
CLRLIN   A 0327   CLRST    A 02D5   CLSCR    A 03E4   CNT0     A 5000   CNT1     A 5001   CNT2     A 5002
CNTM     A 6003   CNWD55   A 1803   CONCL    A 02FD   CRTM     A 1000   CRTS     A 1001   CURAD    A 0FE8
CURSX    A 0FE2   CURSY    A 0FE1   DOWN     A 02AE   ESCP     A 0FEE   ESKAP    A 03A5   ESSQ     A 027B
FMFD     A 03CA   FRAME    A 0167   GD18     A 0359   HOME     A 0397   IN75     A 00F9   INT75    A 1401
KEYDWN   A 0FEA   KEYINP   A 0121   KEYOK    A 0FED   KEYS     A 0131   KPTK     A 0084   KYCHNG   A 01BA
KYLKUP   A 0507   LAST     A 0FD0   LDCUR    A 0388   LEFT     A 036E   LINNUM   A 0019   LINTAB   A 04D5
LNFD     A 03F6   LNFD1    A 03FC   LNGTH    A 0050   LOADX    A 038F   LOC80    A 0FE5   LOOPF    A 00A7
LPKBD    A 0098   NOVER    A 038D   NTOVER   A 0364   OK1      A 049C   OK7      A 015C   ONBOT    A 0453
POPDAT   A 0034   PORTA    A 1800   PORTB    A 1801   PORTC    A 1802   RDKB     A 018F   RETLIN   A 0FF0
RXRDY    A 0113   SAVKEY   A 01AF   SCNLIN   A 0FF1   SCR      A 0211   SETUP    A 010F   SHCON    A 0FEF
STBAUD   A 00DC   STKEY    A 0223   STPTR    A 0FE0   TOPAD    A 0FE3   TPDIS    A 0800   TRANS    A 014B
UP1      A 01B9   UPCUR    A 0333   USCHR    A 0FE7   USTD     A A000   USTP     A A001

ASSEMBLY COMPLETE,  NO ERRORS
```

207780-48

Development Tools 7

7 Development Tools

8051
SOFTWARE PACKAGES

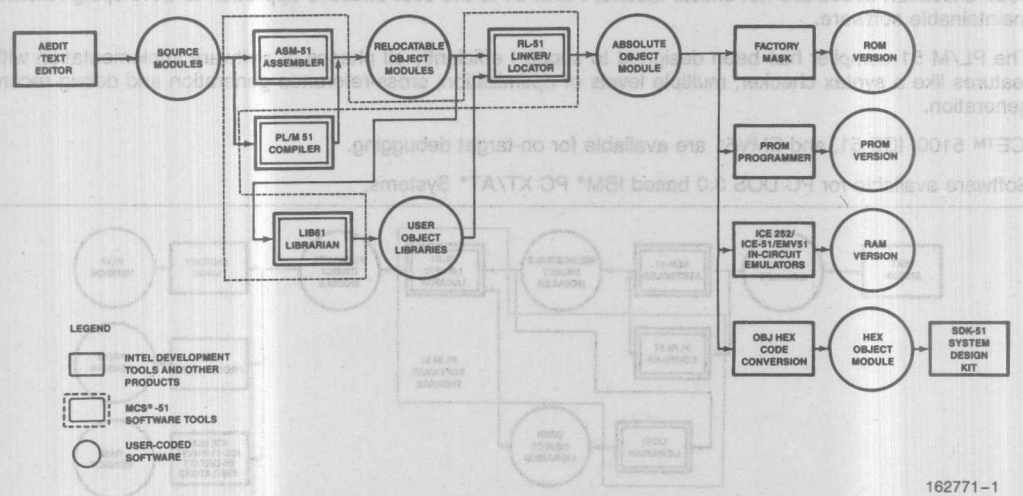
- **Choice of hosts: PCDOS 3.0 based IBM* PC XT/AT*, iRMX®86, iPDS™ System, Series II, Series III, and Series IV**
- **Supports all members of the Intel MCS® -51 architecture**

PL/M51 Software Package Contains the following:

- **PL/M51 Compiler which is designed to support all phases of software implementation**
- **RL51 Linker and Relocator which enables programmers to develop software in a modular fashion**
- **LIB51 Librarian which lets programmers create and maintain libraries of software object modules**

8051 Software Development Package Contains the following:

- **8051 Macro Assembler which gives symbolic access to 8051 hardware features**
- **RL51 Linker and Relocator program which links modules generated by the assembler**
- **LIB51 Librarian which lets programmers create and maintain libraries of software object modules**



162771-1

Figure 1. MCS® -51 Program Development Process

*IBM and AT are registered trademarks of International Business Machine Corporation.

October 1987
Order Number: 162771-005

PL/M 51 SOFTWARE PACKAGE

- **High-level programming language for the Intel MCS® -51 single-chip microcomputer family**
- **Compatible with PL/M 80 assuring MCS® -80/85 design portability**
- **Enhanced to support boolean processing**
- **Tailored to provide an optimum balance among on-chip RAM usage, code size and code execution time**
- **Produces relocatable object code which is linkable to object modules generated by all other 8051 translators**
- **Allows programmer to have complete control of microcomputer resources**
- **Extends high-level language programming advantages to microcontroller software development**
- **Improved reliability, lower maintenance costs, increased programmer productivity and software portability**
- **Includes the linking and relocating utility and the library manager**
- **Supports all members of the Intel MCS® -51 architecture**

PL/M 51 is a structured, high-level programming language for the Intel MCS-51 family of microcomputers. The PL/M 51 language and compiler have been designed to support the unique software development requirements of the single-chip microcomputer environment. The PL/M language has been enhanced to support Boolean processing and efficient access to the microcomputer functions. New compiler controls allow the programmer complete control over what microcomputer resources are used by PL/M programs.

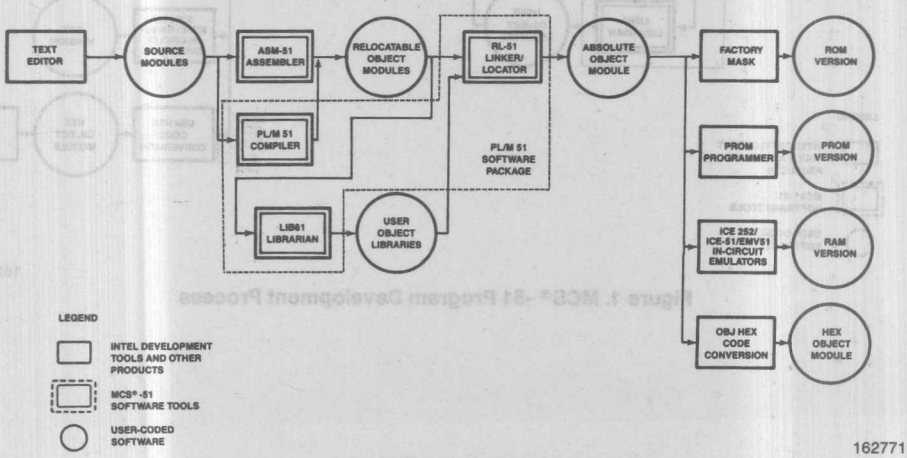
PL/M 51 is largely compatible with PL/M 80 and PL/M 86. A significant proportion of existing PL/M software can be ported to the MCS-51 with modifications to support the MCS-51 architecture. Existing PL/M programmers can start programming for the MCS-51 with a small relearning effort.

PL/M 51 is the high-level alternative to assembly language programming for the MCS-51. When code size and code execution speed are not critical factors, PL/M 51 is the cost-effective approach to developing reliable, maintainable software.

The PL/M 51 compiler has been designed to support efficiently all phases of software implementation with features like a syntax checker, multiple levels of optimization, cross-reference generation and debug record generation.

ICE™ 5100, ICE 51, and EMV51 are available for on-target debugging.

Software available for PC DOS 3.0 based IBM* PC XT/AT* Systems.



162771-2

Figure 2. PL/M51 Software Package

PL/M 51 COMPILER

FEATURES

Major features of the Intel PL/M 51 compiler and programming language include:

Structured Programming

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible, by clearly defining the scope of user variables (local to a private procedure, for example).

Language Compatibility

PL/M 51 object modules are compatible with object modules generated by all other MCS-51 translators. This means that PL/M programs may be linked to programs written in any other MCS-51 language.

Object modules are compatible with In-Circuit Emulators and Emulation Vehicles for MCS-51 processors: the DEBUG compiler control provides these tools with symbolic debugging capabilities.

Supports Three Data Types

PL/M makes use of three data types for various applications. These data types range from one to sixteen bits and facilitate various arithmetic, logic, and address functions:

— Bit: a binary digit
— Byte: 8-bit unsigned number or,
— Word: 16-bit unsigned number.

Another powerful facility allows the use of BASED variables that map more than one variable to the same memory location. This is especially useful for passing parameters, relative and absolute addressing, and memory allocation.

Two Data Structuring Facilities

PL/M 51 supports two data structuring facilities. These add flexibility to the referencing of data stored in large groups.

— Array: Indexed list of same type data elements
— Structure: Named collection of same or different type data elements
— Combinations of Both: Arrays of structures or structures of arrays.

Interrupt Handling

A procedure may be defined with the INTERRUPT attribute. The compiler will generate code to save and restore the processor status, for execution of the user-defined interrupt handler routines.

Compiler Controls

The PL/M 51 compiler offers controls that facilitate such features as:

— Including additional PL/M 51 source files from disk
— Cross-reference
— Corresponding assembly language code in the listing file

Program Addressing Control

The PL/M 51 compiler takes full advantage of program addressing with the ROM (SMALL/MEDIUM/LARGE) control. Programs with less than 2 KB code space can use the SMALL or MEDIUM option to generate optimum addressing instructions. Larger programs can address over the full 64 KB range.

Code Optimization

The PL/M 51 compiler offers four levels of optimization for significantly reducing overall program size.

— Combination or "folding" of constant expressions; "Strength reductions" (a shift left rather than multiply by 2)
— Machine code optimizations; elimination of superfluous branches
— Automatic overlaying of on-chip RAM variables
— Register history: an off-chip variable will not be reloaded if its value is available in a register.

Error Checking

The PL/M 51 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed set of programming and compilation error messages is provided by the compiler and user's guide.

BENEFITS

PL/M 51 is designed to be an efficient, cost-effective solution to the special requirements of MCS-51 Microsystem Software Development, as illustrated by the following benefits of PL/M use:

Low Learning Effort

PL/M 51 is easy to learn and to use, even for the novice programmer.

Earlier Project Completion

Critical projects are completed much earlier than otherwise possible because PL/M 51, a structured high-level language, increases programmer productivity.

Lower Development Cost

Increases in programmer productivity translate immediately into lower software development costs because less programming resources are required for a given programmed function.

Increased Reliability

PL/M 51 is designed to aid in the development of reliable software (PL/M programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.

Easier Enhancements and Maintenance

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

RL51 LINKER AND RELOCATOR

- **Links modules generated by the assembler and the PL/M compiler**
- **Locates the linked object to absolute memory locations**
- **Enables modular programming of software-efficient program development**
- **Modular programs are easy to understand, maintainable and reliable**

The MCS-51 linker and relocator (RL51) is a utility which enables MCS-51 programmers to develop software in a modular fashion. The utility resolves all references between modules and assigns absolute memory locations to all the relocatable segments, combining relocatable partial segments with the same name.

With this utility, software can be developed more quickly because small functional modules are easier to understand, design and test than large programs.

The total number of allowed symbols in user-developed software is very large because the assembler number of symbols' limit applies only per module, not to the entire program. Therefore programs can be more readable and better documented. RL51 can be invoked either manually or through a batch file for improved productivity.

Modules can be saved and used on different programs. Therefore the software investment of the customer is maintained.

RL51 produces two files. The absolute object module file can be directly executed by the MCS-51 family. The listing file shows the results of the link/locate process.

LIB51 LIBRARIAN

The LIB51 utility enables MCS-51 programmers to create and maintain libraries of software object modules. With this utility, the customer can develop standard software modules and place them in libraries, which programs can access through a standard interface. When using object libraries, the linker will call only object modules that are required to satisfy external references.

Consequently, the librarian enables the customer to port and reuse software on different projects—thereby maintaining the customer's software investment.

ORDERING INFORMATION

Order Code	Operating Environment
D86PLM51	PL/M51 Software for PC DOS 3.0 Systems
R86PLM51	PL/M51 Software for iRMX 86 Systems

Documentation Package

PL/M 51 User's Guide

MCS-51 Utilities User's Guide

SUPPORT:

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and monthly Technical Newsletters are available.



8051 SOFTWARE DEVELOPMENT PACKAGE

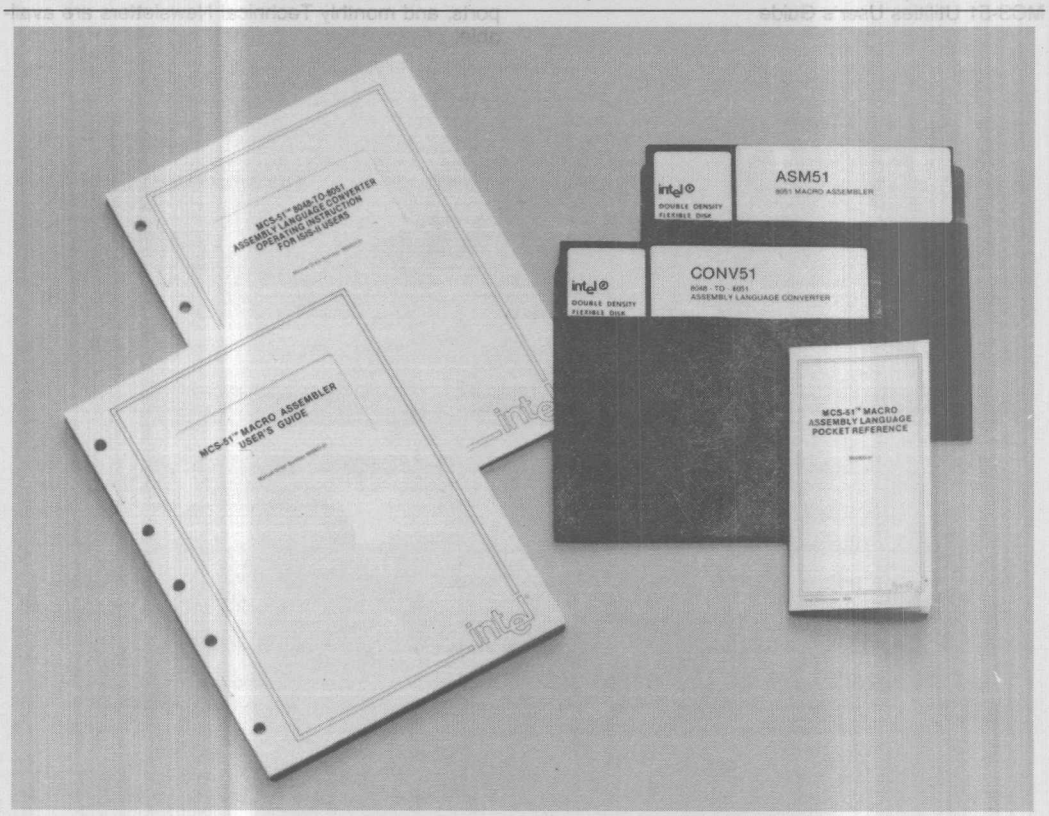
- **Symbolic relocatable assembly language programming for 8051 microcontrollers**
- **Extends Intellec® Microcomputer Development System to support 8051 program development**
- **Produces Relocatable Object Code which is linkable to other 8051 Object Modules**
- **Encourage modular program design for maintainability and reliability**
- **Macro Assembler features conditional assembly and macro capabilities**
- **Supports all members of the Intel MCS® 51 architecture**

The 8051 software development package provides development system support for the powerful 8051 family of single chip microcomputers. The package contains a symbolic macro assembler and relocation/linkage utilities.

The assembler produces relocatable object modules from 8051 macro assembly language instructions. The object code modules can be linked and located to absolute memory locations. This absolute object code may be used to program the 8751 EPROM version of the chip. The assembler output may also be debugged using the new family of ICE 5100 emulators or with the ICE-51™ in-circuit emulator.

The converter translates 8048 assembly language instructions into 8051 source instructions to provide software compatibility between the two families of microcontrollers.

Software available for PC DOS 3.0 based IBM* PC XT/AT Systems.



162771-3

8051 MACRO ASSEMBLER

- **Supports 8051 family program development on Intellec® Microcomputer Development Systems**
- **Gives symbolic access to powerful 8051 hardware features**
- **Produces object file, listing file and error diagnostics**
- **Object files are linkable and locatable**
- **Provides software support for many addressing and data allocation capabilities**
- **Symbolic Assembler supports symbol table, cross-reference, macro capabilities, and conditional assembly**

The 8051 Macro Assembler (ASM51) translates symbolic 8051 macro assembly language modules into linkable and locatable object code modules. Assembly language mnemonics are easier to program and are more readable than binary or hexadecimal machine instructions. By allowing the programmer to give symbolic names to memory locations rather than absolute addresses, software design and debug are performed more quickly and reliably. Furthermore, since modules are linkable and relocatable, the programmer can do his software in modular fashion. This makes programs easy to understand, maintainable and reliable.

The assembler supports macro definitions and calls. This is a convenient way to program a frequently used code sequence only once. The assembler also provides conditional assembly capabilities.

Cross referencing is provided in the symbol table listing, showing the user the lines in which each symbol was defined and referenced.

ASM51 provides symbolic access to the many useful addressing features of the 8051 architecture. These features include referencing for bit and byte locations, and for providing 4-bit operations for BCD arithmetic. The assembler also provides symbolic access to hardware registers, I/O ports, control bits, and RAM addresses. ASM51 can support all members of the 8051 family.

Math routines are enhanced by the MULtiply and DIVide instructions.

If an 8051 program contains errors, the assembler provides a comprehensive set of error diagnostics, which are included in the assembly listing or on another file. Program testing may be performed by using the iUP Universal Programmer and iUP F87/51 personality module to program the 8751 EPROM version of the chip.

ICE 5100, ICE51 and EMV51 are available for program debugging.

RL51 LINKER AND RELOCATOR PROGRAM

- **Links modules generated by the assembler**
- **Locates the linked object to absolute memory locations**
- **Enables modular programming of software for efficient program development**
- **Modular programs are easy to understand, maintainable and reliable**

The 8051 linker and relocator (RL51) is a utility which enables 8051 programmers to develop software in a modular fashion. The linker resolves all references between modules and the relocator assigns absolute memory locations to all the relocatable segments, combining relocatable partial segments with the same name.

With this utility, software can be developed more quickly because small functional modules are easier to understand, design and test than large programs.

The number of symbols in the software is very large because the assembler symbol limit applies only per module not the entire program. Therefore programs can be more readable and better documented.

Modules can be saved and used on different programs. Therefore the software investment of the customer is maintained.

RL51 produces two files. The absolute object module file can be directly executed by the 8051 family. The listing file shows the results of the link/locate process.

LIB51 LIBRARIAN

The LIB51 utility enables MCS-51 programmers to create and maintain libraries of software object modules. With this utility, the customer can develop standard software modules and place them in libraries, which programs can access through a standard interface. When using object libraries, the linker will call only object modules that are required to satisfy external references.

Consequently, the librarian enables the customer to port and reuse software on different projects—thereby maintaining the customer's software investment.

ORDERING INFORMATION

Order Code	Operating Environment
D86ASM51	8051 Assembler for PCDOS 3.0 Systems
R86ASM51	8051 Assembler for iRMX 86 Systems

Documentation Package:

MCS-51 Macro Assembler User's Guide

MCS-51 Utilities User's Guide for 8080/8085 Based Development System

MCS-51 8048-to-8051 Assembly Language Converter Operating Instructions for ISIS-II Users

SUPPORT:

Hotline Telephone Support, Software Performance Reporting (SPR), Software Updates, Technical Reports, Monthly Newsletter available.

8096 SOFTWARE DEVELOPMENT PACKAGES

- **Choice of Hosts**
- **MCS®-96 Software Support Package**
- **C-96/196 Software Package**
- **Supports All Members of the MCS-96 Family**
- **PL/M-96 Software Package**

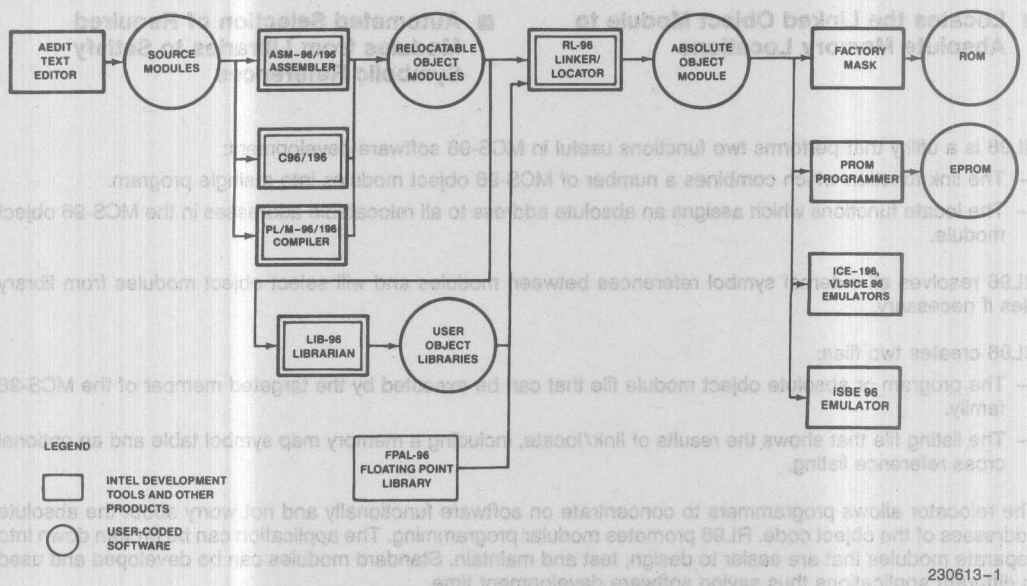
8096 ASSEMBLER PACKAGE

- **Symbolic relocatable assembly language programming for the 8096 microcontroller family**
- **System Utilities for Program Linking and Relocation**
- **Extends Intellec® Microcomputer Development System to support 8096 program development**
- **Encourages modular program design for maintainability and reliability**

The 8096 Software Support Package provides development system support for the 8096 family of 16-bit single chip microcomputers. The support package includes a macro assembler and system utilities.

The assembler produces relocatable object modules from 8096 macro assembly language instructions. The object modules then are linked and located to absolute memory locations.

The assembler and utilities run on PC DOS 3.0 IBM* PC XT/AT Systems.



230613-1

Figure 1. 8096 Software Development Process

*IBM is a registered trademark of International Business Machines Corporation.

October 1987
Order Number: 230613-006

8096 MACRO ASSEMBLER

- **Gives Symbolic Access to Powerful 8096 Hardware Features**
- **Object Files are Linkable and Locatable**
- **Symbolic Assembler Supports Macro Capabilities, Cross Reference, Symbol Table and Conditional Assembly**

ASM-96 is the macro assembler for the MCS family of microcontrollers, including the 80C196. ASM-96 translates symbolic assembly language mnemonics into relocatable object code. Since the object modules are linkable and locatable, ASM-96 encourages modular programming practices.

The macro facility in ASM-96 allows programmers to save development and maintenance time since common code sequences only have to be done once. The assembler also provides conditional assembly capabilities.

ASM-96 supports symbolic access to the many features of the 8096 architecture. An "include" file is provided with all of the 8096 hardware registers defined. Alternatively, the user can define any subset of the 8096 hardware register set.

Math routines are supported with mnemonics for 16 × 16-bit multiply or 32/16-bit divide instructions.

The assembler runs on a PC-DOS 3.0 IBM PC XT/AT.

RL96 LINKER AND RELOCATOR PROGRAM

- **Links Modules Generated by ASM-96, C-96, and PL/M-96**
- **Locates the Linked Object Module to Absolute Memory Locations**
- **Encourages Modular Programming for Faster Program Development**
- **Automated Selection of Required Modules from Libraries to Satisfy Symbolic References**

RL96 is a utility that performs two functions useful in MCS-96 software development:

— The link function which combines a number of MCS-96 object modules into a single program.

— The locate functions which assigns an absolute address to all relocatable addresses in the MCS-96 object module.

RL96 resolves all external symbol references between modules and will select object modules from library files if necessary.

RL96 creates two files:

— The program or absolute object module file that can be executed by the targeted member of the MCS-96 family.

— The listing file that shows the results of link/locate, including a memory map symbol table and an optional cross reference listing.

The relocator allows programmers to concentrate on software functionally and not worry about the absolute addresses of the object code. RL96 promotes modular programming. The application can be broken down into separate modules that are easier to design, test and maintain. Standard modules can be developed and used in different applications thus saving software development time.

FPAL96 FLOATING POINT ARITHMETIC LIBRARY

- **Implements IEEE Floating Point Arithmetic**
- **Basic Arithmetic Operations +, −, ×, /, Mod Plus Square Root**
- **Supports Single Precision 32 Bit Floating Point Variables**
- **Includes an Error Handler Library**

FPAL96 is a library of single precision 32-bit floating point arithmetic functions. All math adheres to the proposed IEEE floating point standard for accuracy and reliability. An error handler to handle exceptions (for example, divide by zero) is included.

The following functions are included:

ADD	NEGATE
SUBTRACT	ABSOLUTE
MULTIPLY	SQUARE ROOT
DIVIDE	INTEGER
COMPARE	REMAINDER

LIB 96

The LIB 96 utility creates and maintains libraries of software object modules. The customer can develop standard modules and place them in libraries. Application programs can then call these modules using predefined interfaces.

LIB 96 uses the following set of commands:

—CREATE:	Creates an empty library file.
—ADD:	Adds object modules to a library file.
—DELETE:	Deletes object modules from a library file.
—LIST:	Lists the modules in the library file.
—EXIT:	Terminates LIB 96

When using object libraries, RL96 will include only those object modules that are required to satisfy external references, thus saving memory space.

ORDERING INFORMATION

Order Code	Operating Environment
D86ASM96	96 Assembler for PC DOS 3.0 Systems

Documentation Package:

MCS-96 Macro Assembler User's Guide
MCS-96 Utilities User's Guide
MCS-96 Assembler and Utilities Pocket Reference Card
8096 Floating Point Arithmetic Library

SUPPORT:

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

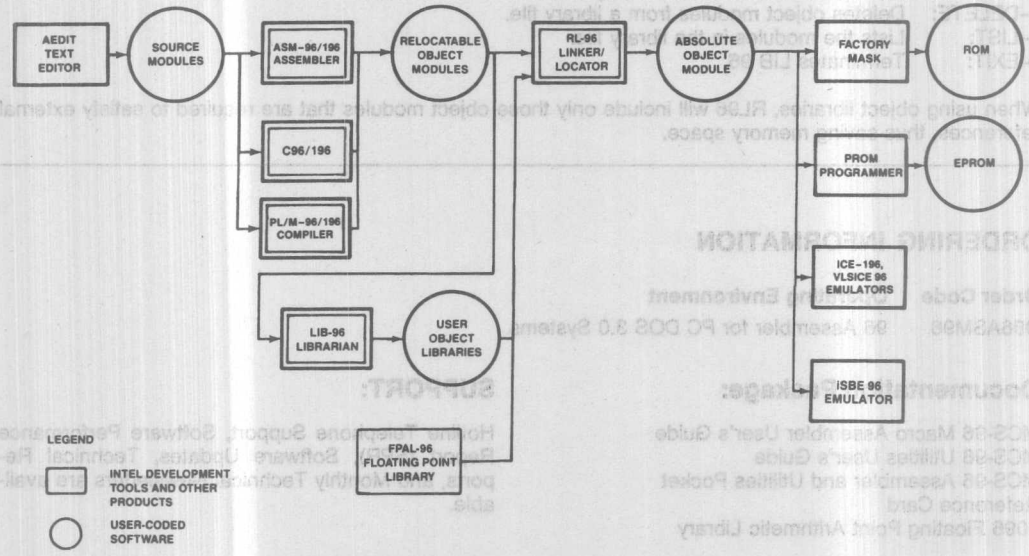
- **Choice of Hosts**
- **Block Structured Language Design Encourages Module Programming**
- **Provides Access to 8096 on Chip Resources**
- **Produces Relocatable Object Code which is Linkable to Object Modules Generated by Other 8096 Translators**
- **Resident on 8086 Intel Microcomputer Development Systems for Higher Performance**
- **Includes a Linking and Relocating Utility and the Library Manager**
- **IEEE Floating Point Library included for Numeric Support**
- **Compatible with PL/M-86 Assuring Design Portability**

PL/M-96 is a structured, high-level programming language useful for developing software for the Intel 8096 family of microcontrollers, including the 80C196. PL/M-96 was designed to support the software requirements of advanced 16 bit microcontrollers. Access to the on chip resources of the 8096 has been provided in PL/M-96.

PL/M-96 is compatible with PL/M-86. Programmers familiar with PL/M will find they can program in PL/M-96 with little relearning effort.

The PL/M-96 compiler translates PL/M-96 high level language statements into 8096 machine instructions. By programming in PL/M an engineer can be more productive in the initial software development cycle of the project. PL/M can also reduce future maintenance and support cost because PL/M programs are easier to understand. PL/M-96 was designed to complement Intel's ASM-96.

PL/M-96 is available for PC DOS 3.0 based IBM PC XT/AT Systems.



230613-1

Figure 2. PL/M-96 Software Package

PL/M-96 COMPILER

FEATURES

Major features of the PL/M-96 compiler and programming language include:

Structured Programming

Programs written in PL/M-96 are developed as a collection of procedures, modules and blocks. Structured programs are easier to understand, maintain and debug. PL/M-96 programs can be made more reliable by clearly defining the scope of user variables (for example, local variables in a procedure). REENTRANT procedures are also supported by PL/M-96.

Language Compatibility

PL/M-96 object modules are compatible with all other object modules generated by Intel MCS-96 translators. Programmers may choose to link ASM-96 and PL/M-96 object modules together.

PL/M-96 object modules were designed to work with other Intel support tools for the MCS-96. The DEBUG compiler control provides these tools with symbolic information.

Data Types Supported

PL/M-96 supports seven data types for programmer flexibility in various logical, arithmetic and addressing functions. The seven data types include:

—BYTE:	8-bit unsigned number
—WORD:	16-bit unsigned number
—DWORD:	32-bit unsigned number
—SHORTINT:	8-bit signed number
—INTEGER:	16-bit signed number
—LONGINT:	32-bit signed number
—REAL:	32-bit floating point number

Another powerful feature are BASED variables. BASED variables allow the user to map more than one variable to the same memory location. This is especially useful for passing parameters, relative and absolute addressing, and memory allocation.

Data Structures Supported

Two data structuring facilities are supported by PL/M-96. The user can organize data into logical groups. This adds flexibility in referencing data.

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of Both: Arrays of structures or structures of arrays

Interrupt Handling

Interrupts are supported in PL/M-96 by defining a procedure with the INTERRUPT attribute. The compiler will generate code to save and restore the program status word when handling hardware interrupts of the MCS-96.

Compiler Controls

Compile time options increase the flexibility of the PL/M-96 compiler. These controls include:

- Optimization
- Conditional compilation
- The inclusion of common PL/M-96 source files from disk
- Cross reference of symbols
- Optional assembly language code in the listing file

Code Optimizations

The PL/M-96 compilers has four levels of optimization for reducing program size.

— Combination of constant expressions; "Strength reductions" (e.g.: a shift left rather than multiply by two)

— Machine code optimizations; elimination of superfluous branches; reuse of duplicate code, removal of unreachable code

— Overlaying of on chip RAM variables

— Optimization of based variable operations

— Use of short jumps where possible

Built in Functions

An extensive list of built in functions has been supplied as part of the PL/M-96 language. Besides TYPE CONVERSION functions, there are built in functions for STRING manipulations. Functions are provided for interrogating the MCS-96 hardware flags such as CARRY and OVERFLOW.

Error Checking

If the PL/M-96 compiler detects a programming or compilation error, a fully detailed error message is provided by the compiler. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This powerful PL/M-96 feature can yield a two times increase in throughput when a user is in the initial program development cycle.

BENEFITS

PLM-96 is designed to be an efficient, cost-effective solution to the special requirements of MCS-96 Microcontroller Software Development, as illustrated by the following benefits of PL/M use:

Low Learning Effort

PL/M-96 is easy to learn and to use, even for the novice programmer.

Earlier Project Completion

Critical projects are completed much earlier than otherwise possible because PL/M-96, a structured high-level language, increases programmer productivity.

Lower Development Cost

Increases in programmer productivity translate immediately into lower software development costs because less programming resources are required for a given programmed function.

Increased Reliability

PL/M-96 is designed to aid in the development of reliable software (PL/M programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status. The more simply the program is stated, the more likely it is to perform its intended function.

Easier Enhancements and Maintainance

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

ORDERING INFORMATION

Order Code	Operating Environment
D86PLM96	PL/M-96 Compiler for PC DOS 3.0 based Systems

Documentation Package

PL/M-96 User's Guide
MCS-96 Utilities User's Guide
MCS-96 Assembler and Utilities Pocket Reference Card
8096 Floating Point Arithmetic Library

SUPPORT

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

C 96 SOFTWARE PACKAGE

- **Implements the Full Programming Capabilities of the C Language**
- **Complies with Draft ANSI Standard**
- **Produces Relocatable Object Code which is Linkable to Object Modules Generated by Other MCS®-96 Translators**
- **Produces High-Density Code That Rivals Assembly in Efficiency**
- **Fully Linkable with the PL/M-96 and ASM-96 Programming Languages**
- **IEEE Floating Point Library (FPAL96) Included for Numeric Support**
- **Supports All of the Standard C Language I/O Library (STDIO)**
- **Includes a Linking and Relocating Utility, an Object-To-Hexadecimal Convertor, and a Library Manager**
- **Supports the 80C196 Architecture**

Intel's C 96 is a general purpose, structured programming language designed to support applications for the 16-bit family of MCS-96 microcontrollers including the 80C196. C 96 implements the C language as described in the Kernighan and Ritchie book, *The C Programming Language* (Prentice-Hall) Software Series, 1978). The latest enhancements to the C programming language as defined by the draft proposed ANSI C standard (e.g., structure assignments, and the void and enum data types) are supported.

The C 96 compiler translates C 96 language statements into MCS-96 machine instructions. The compiler generates code in Intel's relocatable Object Module Format (OMF) without using an intermediate assembly file. The OMF files can then be debugged using either the iSBE-96 emulator, the VLSiCE-96 emulator, or the ICE-196

C 96 is available for the IBM PC AT and the PC XT

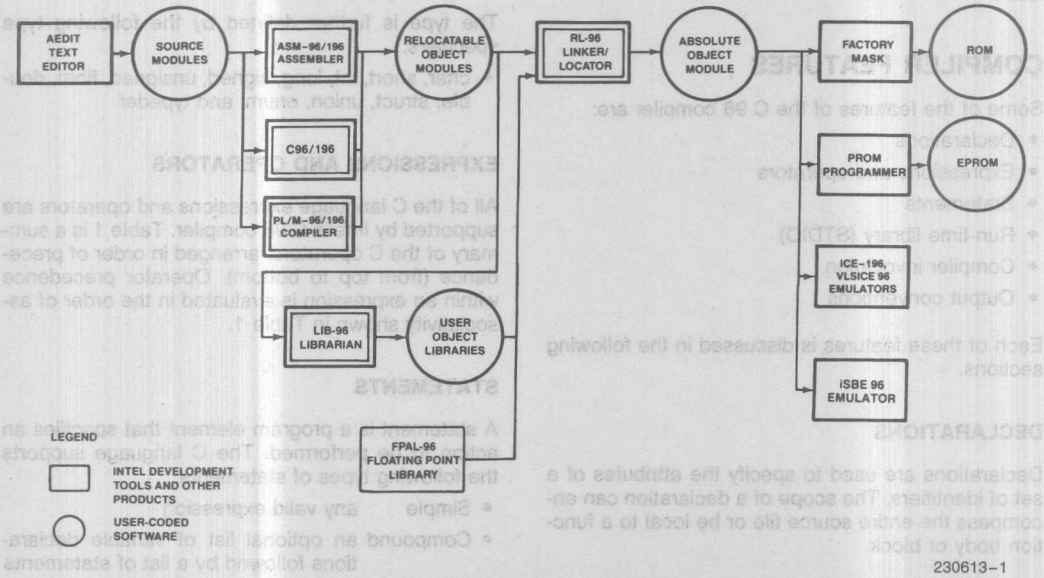


Figure 3. 8096 Software Development Process

C 96 COMPILER

COMPILER DESCRIPTION

Major features of the C 96 compiler include the preprocessor, the parser, and the code generator and optimizer. The code is output in Intel relocatable Object Module Format (OMF). The compiled code can then be debugged with either the iSBE-96 emulator or the VLSiCE-96 emulator.

The preprocessor interprets statements in the source code and performs such actions as macro expansion, file inclusion, and conditional compilation (for example, the #if directive, which specifies optional inclusion or exclusion of code).

The parser performs syntactic and semantic error checking on the code. The code generator converts the parser's output into efficient binary code. The optimizer streamlines the code and generates Intel relocatable OMF code, without creating an intermediate assembly file.

The compiler's DEBUG/NODEBUG control option specifies whether or not the object module should contain debug information. The debug information can be used to debug the compiled program using either the iSBE-96 emulator or the VLSiCE-96 emulator.

COMPILER FEATURES

Some of the features of the C 96 compiler are:

- Declarations
- Expressions and operators
- Statements
- Run-time library (STDIO)
- Compiler invocation
- Output conventions

Each of these features is discussed in the following sections.

DECLARATIONS

Declarations are used to specify the attributes of a set of identifiers. The scope of a declaration can encompass the entire source file or be local to a function body or block.

The storage class specifier defines the location and scope. The storage classes are as follows:

- auto — active block
- extern — external data definition
- static — active data segment or register segment
- typedef — a type definition (not storage allocation) that defines another name or a synonym

The storage class can be further defined with one of the following storage class modifiers:

- const — code segment
- register — machine register
- volatile — I/O port (modifies the extern storage class only)

Identifiers are defined by their type. The types fall into one of the following categories:

- basic — characters, integers, floating point numbers
- derived — arrays, structures, unions, enumerations, functions, and pointers
- void — empty set

The type is further defined by the following type specifiers:

- char, short, int, long, signed, unsigned, float, double, struct, union, enum, and typedef

EXPRESSIONS AND OPERATORS

All of the C language expressions and operators are supported by Intel's C 96 compiler. Table 1 is a summary of the C operators, arranged in order of precedence (from top to bottom). Operator precedence within an expression is evaluated in the order of associativity shown in Table 1.

STATEMENTS

A statement is a program element that specifies an action to be performed. The C language supports the following types of statements:

- Simple — any valid expression
- Compound — an optional list of variable declarations followed by a list of statements

- Selection an if or switch statement which is optionally included dependent on specified conditions
- Iteration a do, while or for statement which executes repeatedly until the controlling value is zero
- Branching a break, continue, goto, or return statement which changes the program control flow

Table 1. Precedence and Associativity

Class	Operator	Associativity
primary	[ ] ( ) . →	left to right
unary	++ -- & * + - ~ sizeof far	right to left
binary mult.	* / %	left to right
binary add	+ -	left to right
binary shift	<< >>	left to right
binary relat.	< > <= >=	left to right
binary equal.	== =	left to right
bitwise AND	&	left to right
bitwise XOR	^	left to right
bitwise OR	|	left to right
logical AND	&&	left to right
logical OR	||	left to right
conditional	? :	right to left
assignment	= *= /= %= += -= <<= + >>= & = ^ = |=	right to left
comma	,	left to right

RUN-TIME LIBRARY (STDIO)

Intel's C 96 compiler supports the standard C language I/O library functions (STDIO). The include files listed in Table 2 are included with the C 96 compiler.

Table 2. C 96 Include Files

Name	Description
ctype.h	Used to declare and map characters.
errno.h	Used for error checking.
setjump.h	Used to bypass a normal call/return.
stdio.h	Used for standard I/O functions.
string.h	Used to manipulate strings.
time.h	Used to manipulate the time and date.

Character and arithmetic conversion functions are also included (atof, atoi, atol, cstr, tolower, toupper, and udistr).

COMPILER INVOCATION

Intel's C 96 compiler is invoked with the following general syntax:

c96 *pathname* [*controls*]

The following invocation controls are some of the options supported by the C 96 compiler.

- Object file controls—DEBUG/NODEBUG, OBJECT, OPTIMIZE (0 through 3), REGISTERS, REGOVERLAY/NOREGOVERLAY, TYPE/NOTYPE
- Listing controls (selection and content)—CODE/NOCODE, COND/NOCOND, LIST/NOLIST, LISTINCLUDE/NOLISTINCLUDE, PREPRINT/NOPREPRINT, SYMBOLS/NOSYMBOLS, XREF/NOXREF
- Listing format controls—PAGING/NOPAGING, PAGELENGTH, PAGEWIDTH
- Source inclusion control—INCLUDE

The REENTRANT/NOREENTRANT extension has been added to the C 96 compiler invocation controls to enhance the compiler's use of the MCS-96 architecture. This extension enables the compiler to fully use the large register set of the MCS-96 family of microprocessors. When porting to programs in other environments, these keywords should be either removed or defined as null.

Output Conventions

The C 96 compiler produces a listing file and an object file. The listing file contains a formatted list of the source code and a list of compiler error messages. The compiler produces the object file in Intel's relocatable OMF code directly, without creating an intermediate assembly file.

BENEFITS

There are many benefits to the C 96 compiler, as explained in the following sections.

PROGRAM DEBUGGING

With the DEBUG control the C 96 compiler produces extensive debug information, including symbols. The debug information can be used to debug the program code with either the VLSiCE-96 emulator or the iSBE-96 emulator. This serves to enhance programmer productivity.

FASTER COMPILATION

The C 96 compiler creates Intel object module format (OMF) directly, without creating an intermediate assembly file. This increases the compiler's execution speed.

PORTABLE CODE

Code portability has been designed into the C 96 compiler. The C 96 code is fully linkable with both the PL/M-96 and the ASM-96 programming languages.

Because the compiler supports the standard C library and produces Intel OMF code, programs developed on a variety of machines can be transported to the MCS-96. In addition, because C 96 conforms to accepted C language standards, programmers can quickly begin programming the MCS-96.

FULL MANIPULATION OF THE 8096 MICROCONTROLLER

The C 96 compiler has been highly optimized for the MCS-96 architecture. The REENTRANT/NOREENTRANT control has been added so that the compiler can identify non-reentrant procedures. This is extremely useful because it enables the programmer to have full access to the large MCS-96 register set.

With the C 96 compiler, the programmer can declare register variables that are not local to any procedure. Due to the large register set of the MCS-96 architecture, the compiler can dedicate registers to such variables.

SOFTWARE SUPPORT

Intel's Software Support Service provides maintenance on software packages with software support contracts which include subscription services, information phone support, and updates. Consulting services can be arranged for on-site assistance at the customer's location for both short-term and long-term needs. For more information, contact your local Intel Sales Office.

ORDERING INFORMATION

Part Number	Description
D86C96	C 96 Software Package PL/M-96 packages also include the RL96 Linker and Relocator, the FPAL96 Floating Point Library, and the LIB96 librarian utility.

Operating Environment

IBM PC AT
IBM PC XT

8086/80186 SOFTWARE PACKAGES

8086/80186 Software Development Package

- Macro Assembler with Complete System Development Capability for 8086/80186 Designs
- Complete Set of Utilities for Object Module Management and Program Linkage

FORTRAN 8086/80186 Software Package

- Features High-Level Language Support for Floating-Point Calculation, Transcendentals, Interrupt Procedures, and Run-Time Exception Handling
- Meets ANSI FORTRAN 77 Subset Language Specifications
- Supports Complex Data Types

PASCAL 8086/80186 Software Package

- Object Compatible and Linkable with PL/M 8086, ASM 8086 and FORTRAN 86
- Supports Large Array Operation

PL/M 8086/80186 Software Package

- Advanced Structured System Implementation Language for Algorithm Development
- Easy-to-Learn Block-Structured Language Encourages Program Modularity

iC-86 Compiler for the 8086/80186

- Implements Full ANSI Standard C Language
- Produces High Density Code Rivaling Assembler

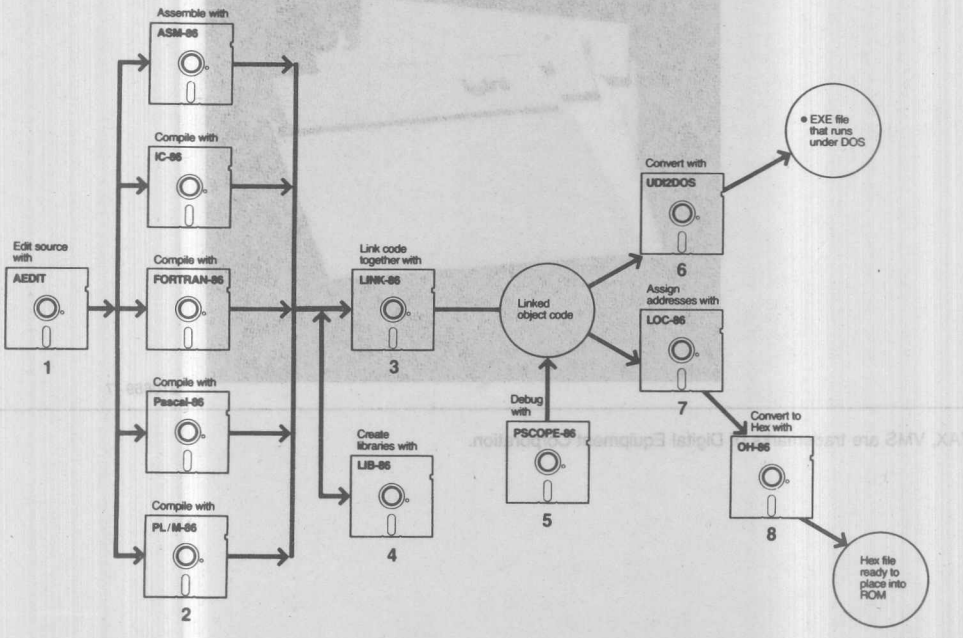


Figure 1. Program modules compiled with any of the 8086 languages may be linked together. Each language is compatible with Intel's debug tools. This is an example of development under DOS.

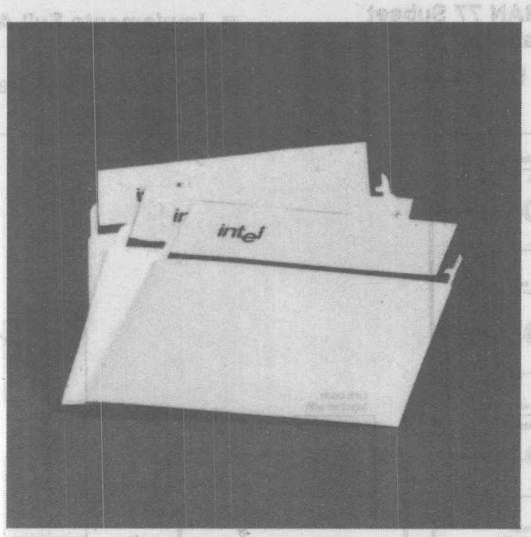
October 1987
Order Number: 210689-008

8086/80186 SOFTWARE DEVELOPMENT PACKAGE

- **Complete System Development Capability for High-Performance 8086 Applications**
- **Macro Assembler for Machine-Level Programming**
- **System Utilities for Program Linkage and Relocation**
- **Package Supports Program Development with PLM-86, Pascal-86, FORTAN 86, & iC 86**
- **Available on a Choice of Hosts**

The 8086 Software Development package contains a macro assembler, a program linker (for linking separately compiled modules together, a system locator, library manager, an object to hex code converter, and a conversion utility to create DOS executable files.

All the utilities in the Software Development Package run on the Intel Microcomputer Development Systems (Series III/Series IV) as well as the IBM PC XT/AT DEC VAX† Minicomputer under the VMS† Operating System, and Intel systems 86/3XX under iRMX™86, and Intel System 286/3XX under iRMX™286.



210689-7

†VAX, VMS are trademarks of Digital Equipment Corporation.

8086/80186 MACRO ASSEMBLER

- **Produces Relocatable Object Code Which is Linkable to All Other Intel 86/186 Object Modules, Generated by Intel 8086 Compilers**
- **Powerful and Flexible Text Macro Facility with Three Macro Listings Options to Aid Debugging**
- **Highly Mnemonic and Compact Language, Most Mnemonics Represent Several Distinct Machine Instructions**
- **"Strongly Typed' Assembler Helps Detect Errors at Assembly Time**
- **High-Level Data Structuring Facilities Such as "STRUCTURES" and "RECORDS"**
- **Over 120 Detailed and Fully Documented Error Messages**

ASM-86 is the "high-level" macro assembler for the 86/186 assembly language. ASM-86 translates symbolic 86/186 assembly language mnemonics into 86/186 relocatable object code.

ASM-86 should be used where maximum code efficiency and hardware control is needed. The 86/186 assembly language includes approximately 100 instruction mnemonics. From these few mnemonics the assembler can generate over 3,800 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 100 mnemonics to generate all possible 86/186 machine instructions. ASM-86 will generate the shortest machine instruction possible given no forward referencing or given explicit information as to the characteristics of forward referenced symbols.

ASM-86 offers many features normally found only in high-level languages. The 86/186 assembly language is strongly typed. The assembler performs extensive checks on the usage of variables and labels. The assembler uses the attributes which are derived explicitly when a variable or label is first defined, then makes sure that each use of the symbol in later instructions conforms to the usage defined for that symbol. This means that many programming errors will be detected when the program is assembled, long before it is being debugged on hardware.

LINK-86

- **Automatic Combination of 8086 Programs Separately Translated Using Intel Compilers or Assemblers into Relocatable Object Module**
- **Automatic Selection of Required Modules from Specified Libraries to Satisfy Symbolic References**
- **Extensive Debug Symbol Manipulation, allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**
- **Automatic Generation of a Summary Map Giving Results of the LINK-86 Process**
- **Abbreviated Control Syntax**
- **Relocatable Modules May Be Merged into a Single Module Suitable for Inclusion in a Library**
- **Supports "Incremental" Linking**
- **Supports Type Checking of Public and External Symbols**

LINK-86 combines object modules specified in the LINK-86 input list into a single output module. LINK-86 combines segments from the input modules according to the order in which the modules are listed.

LINK-86 will accept libraries and object modules built from any Intel translator generating 8086 Relocatable Object Modules.

Support for incremental linking is provided since an output module produced by LINK-86 can be an input to another link. At each stage in the incremental linking process, unneeded public symbols may be purged.

LINK-86 supports type checking of PUBLIC and EXTERNAL symbols reporting a warning if their types are not consistant.

LINK-86 will link any valid set of input modules without any controls. However, controls are available to control the output of diagnostic information in the LINK-86 process and to control the content of the output module.

LINK-86 allows the user to create a large program as the combination of several smaller, separately compiled modules. After development and debugging of these component modules the user can link them together, locate them using LOC-86 and enter final testing with much of the work accomplished.

- **Automatic Generation of a Summary Map Giving Starting Address, Segment Addresses and Length, and Debug Symbols and Their Addresses**
- **Abbreviated Control Syntax**
- **Segments May be Relocated to Best Match Users Memory Configuration**
- **Extensive Debug Symbol Manipulation Allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**

Relocatability allows the programmer to code programs or sections of programs without having to know the final arrangement of the object code in memory.

LOC-86 converts relative addresses in an input module in 86/186 object module format to absolute addresses. LOC-86 orders the segments in the input module and assigns absolute addresses to the segments. The sequence in which the segments in the input module are assigned absolute addresses is determined by their order in the input module and the controls supplied with the command.

LOC-86 will relocate any valid input module without any controls. However, controls are available to control the output of diagnostic information in the LOC-86 process, to control the content of the output module, or both.

The program you are developing will almost certainly use some mix of random access memory (RAM), read-only memory (ROM), and/or programmable read-only memory (PROM). Therefore, the location of your program affects both cost and performance in your application. The relocation feature allows you to develop your program and then simply relocate the object code to suit your application.

LIB-86

- **LIB-86 is a Library Manager Program which Allows You to:**
  - **— Create Specifically Formatted Files to Contain Libraries of Object Modules**
  - **— Maintain These Libraries by Adding or Deleting Modules**
  - **— Print a Listing of the Modules and Public Symbols in a Library File**
- **Libraries Can be Used as Input to LINK-86 which Will Automatically Link Modules from the Library that Satisfy External References in the Modules Being Linked**
- **Abbreviated Control Syntax**

Libraries aid in the job of building programs. The library manager program LIB-86 creates and maintains files containing object modules. The operation of LIB-86 is controlled by commands to indicate which operation LIB-86 is to perform. The commands are:

CREATE: creates an empty library file
ADD: adds object modules to a library file
DELETE: deletes modules from a library file
LIST: lists the module directory of library files
EXIT: terminates the LIB-86 program and returns control to VMS

When using object libraries, the linker will call only those object modules that are required to satisfy external references, thus saving memory space.

OH-86

- **Converts an 86/186 Absolute Object Module to Symbolic Hexadecimal Format**
- **Facilitates Preparing a File for Loading by Symbolic Hexadecimal Loader (e.g. iSBC™ Monitor SDK-86 Loader), or Universal PROM Mapper**
- **Converts an Absolute Module to a More Readable Format that can be Displayed on a CRT or Printed for Debugging**

The OH-86 utility converts an 86/186 absolute object module to the hexadecimal format. This conversion may be necessary for later loading by a hexadecimal loader such as the iSBC 86/12 monitor or the Universal PROM Mapper. The conversion may also be made to put the module in a more readable format that can be displayed or printed.

The module to be converted must be in absolute form; the output from LOC-86 is in absolute format.

SPECIFICATIONS

Documentation Package

ASM-86 Assembly Language Reference Manual

8086/87/88 Macro Assembler Operating Instructions

iAPX 86 Family Utilities User's Guide

Support Available

Software Updates, Subscription Service, Hotline Support

ORDERING INFORMATION

Order Code	Operating Environment
D86ASM86	IBM PC XT/AT running PC DOS Version 3.0 or later
VVSASM86	VAX†/VMS†
MVVSASM86	MICROVAX†/VMS†
R86ASM86	Intel 86/3XX Systems running: iRMX™ 86
R286ASM286	Intel 286/3XX Systems running: iRMX™ 286

†MICROVAX, VAX, VMS are trademarks of Digital Equipment Corporation.

*IBM, AT are registered trademarks of International Business Machines Corporation.

FORTRAN 8086/80186 SOFTWARE PACKAGE

- **Features High-Level Language Support for Floating-Point Calculations, Transcendentals, Interrupt Procedures, and Run-Time Exception Handling**
- **Meets ANSI FORTRAN 77 Subset Language Specifications**
- **Supports 8086/20, 8088/20 Numeric Data Processor for Fast and Efficient Execution of Numeric Instructions**
- **Uses REALMATH Floating-Point Standard for Consistent and Reliable Results**
- **Supports Arrays Larger Than 64K**
- **Unlimited User Program Symbols**
- **Offers Upward Compatibility with FORTRAN 80**
- **Provides FORTRAN Run-Time Support for 86/186 Based Design**
- **Provides Users Ability to do Formatted and Unformatted I/O with Sequential or Direct Access Methods**
- **I^2ICE^{TM} Symbolic Debugging Fully Supported**
- **PSCOPE Source Level Debugging Fully Supported**
- **Supports Complex Data Types**
- **Choice of Industry Standard Hosts**

FORTRAN 86/186 meets the ANSI FORTRAN 77 Language Subset Specification and includes many features of the full standard. Therefore, the user is assured of portability of most existing ANS FORTRAN programs and of full portability from other computer systems with an ANS FORTRAN 77 Compiler.

FORTRAN 86/186 is available to run on the Intel Microcomputer Development Systems (Series III/Series IV) as well as the IBM PC XT/AT running PC DOS Version 3.0 or later, Digital Equipment VAX†/VMS† and Intel System 86/3XX running iRMX™ 86 operating system.

FORTRAN 86/186 is one of a complete family of compatible programming languages for 8086, 8088, 80186, 80188 development: PL/M, Pascal, FORTRAN, C, and Assembler. Therefore, users may choose the language best suited for a specific problem solution.

---

†VAX, VMS are trademarks of Digital Equipment Corporation.

*IBM, AT are registered trademarks of International Business Machines Corporation.

FEATURES

Extensive High-Level Language Numeric Processing Support

Single (32-bit), double (64-bit), and double extended precision (80-bit) complex (two 32-bit), and double complex (two 64-bit) floating-point data types

REALMATH Proposed IEEE Floating-Point Standard) for consistent and reliable results

Full support for all other data types: integer, logical, character

Ability to use hardware (8086/20, 8088/20 Numeric Data Processor) or software (simulator) floating-point support chosen at link time

ANSI FORTRAN 77 Standard

Intel® Microprocessor Support

FORTRAN 86/186 language features support of 8086/20, 8088/20 Numeric Data Processor

Compiler generates in-line iAPX 8086/20, 8088/20 Numeric Data Processor object code for floating-point arithmetic (See Figure 2)

Intrinsics allow user to control iAPX 8086/20, 8088/20 Numeric Data processor

8086, 8088, 80186, 80188 architectural advantages used for indexing and character-string handling

Symbolic debugging of application using ICE emulators

Source level debugging using PSCOPE

FLOATING-POINT-STATEMENT

```
      TEMPER = (PRESS - VOLUM / QUEK) - 3.45 / (PRESS - VOLUM / QUEK
     & - (PRESS - VOLUM / QUEK) * (PRESS - VOLUM / QUEK)
```

OBJECT CODE GENERATED

```
Intel FORTRAN 8086 Compiler

                                                        ; STATEMENT # 2
0013    9BD9060C00          FLD         VOLUM
0018    9BD8360000          FDIV        QUEK
001D    9BD82E0800          FSUBR       PRESS
0022    9BDDD1              FST         TOS+1H
0025    9B2ED83E0000        FDIVR       CS:@CONST
002B    9BD9C9              FXCHG       TOS+1H
002E    9BDDD2              FST         TOS+2H
0031    9BDEE9              FSUBRP
0034    9BD9C1              FLD         TOS+1H
0037    9BD8C8              FMUL        TOS
003A    9BDDC2              FFREE       TOS+2H
003D    9BDEE1              FSUBP
0040    9BD91E0400          FSTP        TEMPER
0045    9B                  WAIT
```

8086/20, 8088/20 MACHINE-CODE

ASSEMBLER MNEMONICS

Figure 2. Object code generated by FORTRAN 86/186 for a floating-point calculation using 8086/20, 8088/20 Numeric Processor.

Microprocessor Application Support

— Direct byte- or word-oriented port I/O
— Reentrant procedures
— Interrupt procedures

BENEFITS

FORTRAN 86/186 provides a means of developing application software for the Intel 86/186 products lines in a familiar, widely accepted, and industry-standard programming language. FORTRAN 86/186 will greatly enhance the user's ability to provide cost-effective software development for Intel microprocessors as illustrated by the following:

Early Project Completion

FORTRAN is an industry-standard, high-level numerics processing language. FORTRAN programmers can use FORTRAN 86/186 on microprocessor projects with little retraining. Existing FORTRAN software can be compiled with FORTRAN 86/186 and programs developed in FORTRAN 86/186 can run on other computers with ANSI FORTRAN 77 with little or no change. Libraries of mathematical programs using ANSI 77 standards may be compiled with FORTRAN 86/186.

Application Object Code Portability for a Processor Family

FORTRAN 86/186 modules "talk" to the resident Intellec development operating system using Intel's standard interface for all development-system software. This allows an application developed under the ISIS-II operating system to execute on iRMX/86, or a user-supplied operating system by linking in the iRMX/86 or other appropriate interface library. A standard logical-record interface enables communication with non-standard I/O devices.

Comprehensive, Reliable and Efficient Numeric Processing

The unique combination of FORTRAN 8086/8088, 8086/20, 8088/20 Numeric Data processor, and REALMATH (Proposed IEEE Floating-Point Standard) provide universal consistency in results of numeric computations and efficient object code generation.

SPECIFICATIONS

Documentation Package

FORTRAN 86/88/186/188 User's Guide

ORDERING INFORMATION

Order Code	Operating Environment
D86FOR86	IBM PC XT/AT running PC DOS Version 3.0 or later
R86FOR86	Intel System 86/3XX running iRMX 86
VVSFOR86	For 86 VAX/VMS 4.3 and later

SUPPORT AVAILABLE

Software updates, Subscription Service, Hotline Support.

PASCAL 86/186 SOFTWARE PACKAGE

- **Choice of Industry Standard Hosts**
- **Object Compatible and Linkable with PL/M 86/186, ASM 86/186, iC86/186 and FORTRAN 86/186**
- **I²ICE™ Symbolic Debugging Fully Supported**
- **PSCOPE Source Level Dubugging Fully Supported**
- **Implements REALMATH for Consistent and Reliable Results**
- **Supports Large Array Operation**
- **Unlimited User Program Symbols**
- **Supports 8086/20, 8088/20 Numeric Data Processors**
- **Strict Implementation of ISO Standard Pascal**
- **Useful Extensions Essential for Microcomputer Applications**
- **Separate Compilation with Type-Checking Enforced Between Pascal Modules**
- **Compiler Option to Support Full Run-Time Range-Checking**

PASCAL 86/186 conforms to and implements the ISO PASCAL standard. The language is enhanced to support microcomputer applications with special features, such as separate compilation, interrupt handling and direct port I/O. To assist the development of portable software, the compiler can be directed to flag all non-standard features.

The PASCAL 86/186 compiler runs on Series III and Series IV Microcomputer Development Systems, as well as the IBM* XT/AT* running PC DOS Version 3.0 or later, Digital Equipment VAX/VMS†, and Intel System 8086/3XX running iRMX™ 86.

A well-defined I/O interface is provided for run-time support. This allows a user-written operating system to support application programs as an alternate to the development system environment. Program modules compiled under PASCAL 86/186 are compatible and linkable with modules written in PL/M 86/186, ASM 86/186, C86/186 or FORTRAN 86/186. With a complete family of compatible programming languages for the 86/186 one can implement each module in the language most appropriate to the task at hand.

PASCAL 86/186 object modules contain symbol and type information for program debugging using ICE emulators and PSCOPE source language debugger. For final production version, the compiler can remove this extra information and code.

---

†VAX, VMS are trademarks of Digital Equipment Corporation.

FEATURES

Includes all the language features of Jensen & Wirth Pascal as defined in the ISO Pascal Standard.

Supports required extensions for microcomputer applications.

— Interrupt handling

— Direct port I/O

Separate compilation extensions allow:

— Modular decomposition of large programs

— Linkage with other Pascal modules as well as PL/M 86/186, ASM 86/186, C86/186 and FORTRAN 86/186

— Enforcement of type-checking at LINK-time

Supports numerous compiler options to control the compilation process, to INCLUDE files, flag non-standard Pascal statements and others to control program listing and object modules.

Utilizes the IEEE standard for Floating-Point Arithmetic (the Intel REALMATH standard) for arithmetic operations.

Well-defined and documented run-time operating system interfaces allow the user to execute the applications under user-designed operations systems.

Predefined type extensions allow:

— Create precision in read, integer, and unsigned calculations.

— Means to check 8087 errors

— Circumvention of rigid type checking on calls to non-Pascal routines

BENEFITS

Provides a standard Pascal for 86/186 based applications.

— Pascal has gained wide acceptance as a portable application language for microcomputer applications

— It is being taught in many colleges and universities around the world

— It is easy to learn, originally intended as a vehicle for teaching computer programming

— Improves maintainability: Type mechanism is both strictly enforced and user extendable

— Few machine specific language constructs

Strict implementation of the proposed ISO standard for Pascal aids portability of application programs. A compile time option checks conformance to the standard making it easy to write conforming programs.

PASCAL 86/186 extensions via predefined procedures for interrupt handling and direct port I/O make it possible to code an entire application in Pascal without compromising portability.

Standard Intel REALMATH is easy to use and provides reliable results, consistent with other Intel languages and other implementations of the IEEE proposed Floating-Point standard.

Provides run-time support for co-processors. All real-type arithmetic is performed on the 86/20 numeric data processor unit or software emulator. Run-time library routines, common between Pascal and other Intel languages (such as FORTRAN), permit efficient and consistently accurate results.

Extended relocation and linkage support allows the user to link Pascal program modules with routines written in other languages for certain parts of the program. For example, real-time or hardware dependent routines written in ASM 86/186 or PL/M 86/186 can be linked to Pascal routines, further extending the user's ability to write structured and modular programs.

PASCAL 86/186 programs "talk" to the resident operating system using Intel's standard interface for translated programs. This allows users to replace the development operating system by their own operating systems in the final application.

PASCAL 8086/8088 takes full advantage of 86/186 high level language architecture to generate efficient machine code.

Compiler options can be used to control the program listings and object modules. While debugging, the user may generate additional information such as the symbol record information required and useful for debugging using PSCOPE or ICE emulation. After debugging, the production version may be streamlined by removing this additional information.

SPECIFICATIONS

ORDERING INFORMATION

Ordering Code	Operating Environment
D86PAS86	IBM PC XT/AT running PC DOS Version 3.0 or later
R86PAS86	Intel System 86/3XX running iRMX™ 86
VVSPAS86	VAX/VMS
MVVPAS86	MICROVAX/VMS

Documentation Package

PASCAL 86 User's Guide

SUPPORT

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

PL/M 86/186 Software Package

- **Systems Programming Language for the 86/186 Processors**
- **Language is Upward Compatible from PL/M 80, Assuring MCS®-80/85 Design Portability**
- **Advanced Structured System Implementation Language for Algorithm Development**
- **Supports 16-Bit Signed Integer and 32-Bit Floating Point Arithmetic in Accordance with IEEE Proposed Standard**
- **Easy-to-Learn Block-Structured Language Encourages Program Modularity**
- **Improved Compiler Performance Now Supports More User Symbols and Faster Compilation Speeds**
- **Produces Relocatable Object Code Which Is Linkable to All Other 8086 Object Modules**
- **Code Optimization Assures Efficient Code Generation and Minimum Application Memory Utilization**
- **Built-In Syntax Checker Doubles Performance for Compiling Programs Containing Errors**
- **Resident on Choice of Hosts**
- **I^2ICE Symbolic Debugging Fully Supported**
- **PSCOPE Source Level Debugging Fully Supported**

PL/M 86/186 is an advanced, structured, high-level systems programming language. The PL/M 86/186 compiler was created specifically for performing software development for the Intel 86/186 Microprocessors. PL/M was designed so that program statements naturally express the program algorithm. This frees the programmer to concentrate on the logic of the program without concern for burdensome details of machine or assembly language programming (such as register allocation, meanings of assembler mnemonics, etc.).

The PL/M 86/186 compiler efficiently converts free-form PL/M language statements into machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

The use of PL/M high-level language for system programming, instead of assembly language, results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-up maintenance costs for the user.

PL/M 8086 is available to run on the Intellec® Microcomputer Development Systems (Series III/Series IV) as well as the IBM PC XT/AT, DEC VAX†/VMS†, and Intel System 8086/3XX running iRMX™ 86.

---

†VAX, VMS are trademarks of Digital Equipment Corporation.

FEATURES

Major features of the Intel PL/M 8086 compiler and programming language include:

Block Structure

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible, by clearly defining the scope of user variables (local to a private procedure).

The use of procedures to break down a large problem is paramount to productive software development. The PL/M 8086 implementation of a block structure allows the use of REENTRANT (recursive) procedures, which are especially useful in system design.

Language Compatibility

PL/M 8086 object modules are compatible with object modules generated by all other 8086 translators. This means that PL/M programs may be linked to programs written in any other 8086 language.

Object modules are compatible with In-Circuit Emulators; DEBUG compiler control provides the In-Circuit Emulators with symbolic debugging capabilities.

PL/M 8086 Language is upward compatible with PL/M 80, so that application programs may be easily ported to run on the 8086.

Supports Seven Data Types

PL/M makes use of seven data types for various applications. These data types range from one to four bytes, and facilitate various arithmetic, logic, and addressing functions:

— Byte: 8-bit unsigned number
— Word: 16-bit unsigned number
— DWORD: 32-bit unsigned number
— Integer: 16-bit signed number
— Read: 32-bit floating point number
— Pointer: 16-bit or 32-bit memory address indicator
— Selector: 16-bit base portion of a pointer

Another powerful facility allows the use of BASED variables that map more than one variable to the same memory location. This is especially useful for passing parameters, relative and absolute addressing, and memory allocation.

Two Data Structuring Facilities

In addition to the five data types and based variables, PL/M supports two data structuring facilities. These help the user to organize data into logical groups.

— Array: Indexed list of same type of data elements
— Structure: Named collection of same or different type data elements
— Combinations of Each: Arrays of structures or structures of arrays

8087 Numerics Support

PL/M programs that use 32-bit REAL data may be executed using the Numeric Data Processor for improved performance. All floating-point operations supported by PL/M may be executed on the 8086/20 or 8088/20 NDP, or the 8087 Emulator (a software module) provided with the package. Determination of use of the chip or Emulator takes place at linktime, allowing compilations to be run-time independent.

Built-In String Handling Facilities

The PL/M 8086 language contains built-in functions for string manipulation. These byte and word functions perform the following operations on character strings: MOVE, COMPARE, TRANSLATE, SEARCH, SKIP, and SET.

Interrupt Handling

PL/M has the facility for handling interrupts. A procedure may be defined with the INTERRUPT attribute, and the compiler willl automatically initialize an interrupt vector at the appropriate memory location. The compiler will also generate code to save and restore the processor status, for execution of the user-defined interrupt handler routine. The procedure SET$INTERRUPT, the function retuning an INTERRUPT$PTR, and the PL/M statement CAUSE$INTERRUPT all add flexibility to user programs involving interrupt and handling.

Compiler Controls

Including several that have been mentioned, the PL/M 8086 compiler offers more than 25 controls that facilitate such features as:

— Conditional compilation

— Including additional PL/M source files from disk

— Corresponding assembly language code in the listing file

— Setting overflow conditions for run-time handling

Segmentation Control

The PL/M 8086 compiler takes full advantage of program addressing with the SMALL, COMPACT, MEDIUM, and LARGE segmentation controls. Programs with less than 64 KB total code space can exploit the most efficient memory addressing schemes, which lowers total memory requirements. Larger programs can exploit the flexibility of extended one-megabyte addressing.

Code Optimization

The PL/M 8086 compiler offers four levels of optimization for significantly reducing overall program size.

— Combination or "folding" of constant expressions; and short-circuit evaluation of Boolean expressions

— "Strength reductions" (such as a shift left rather than multiply by 2); and elimination of common sub-expressions within the same block

— Machine code optimizations; elimination of superfluous branches; re-use of duplicate code; removal of unreachable code

— Byte comparisons (rather than 20-bit address calculations) for pointer variables; optimization of based-variable operations

Error Checking

The PL/M 8086 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed set of programming and compilation errors is provided by the compiler.

```
M:DO; /* Beginning of module */
      SORTPROC: PROCEDURE (PTR, COUNT, RECSIZE, KEYINDEX) PUBLIC;
                DECLARE PTR POINTER, (COUNT, RECSIZE, KEYINDEX) INTEGER,

      /* Parameters:
         PTR is pointer to first record.
         COUNT is number of records to be sorted.
         RECSIZE is number of bytes in each record—max is 128.
         KEYINDEX is byte position within each record of a BYTE scalar
                  to be used as sort key. */
                DECLARE RECORD BASED PTR (1) BYTE,
                        CURRENT (128) BYTE,
                        (I, J) INTEGER;

      SORT:     DO J=1 TO COUNT-1;
                   CALL MOVB(@RECORD(J*RECSIZE), @CURRENT, RECSIZE);
                   I=J;
      FIND:        DO WHILE I>0
                      AND RECORD((I-1)*RECSIZE+KEYINDEX)
                      >CURRENT(KEYINDEX);
                   CALL MOVB(@RECORD((I-1)*RECSIZE),
                             @RECORD(I*RECSIZE),
                             RECSIZE);
                   I=I-1;
                END FIND;
                CALL MOVB(@CURRENT, @RECORD(I*RECSIZE), RECSIZE);
             END SORT;
          END SORTPROC;
END M:    /*End of module*/
```

PUBLIC and EXTERNAL attributes promote program modularity.

"Based" Variables allow manipulation of external data by passing the base of the data structure (a pointer). This minimizes the STACK space used for parameter passing, and the execution time to perform many STACK operations.

The "AT" operator returns the *address* of a variable, instead of its contents. This is very useful in passing pointers for based variables.

One of several PL/M built-in procedures for string manipulation.

210689-5

Figure 3. Sample PL/M 8086 Program

BENEFITS

PL/M 8086 is designed to be an efficient, cost-effective solution to the special requirements of 8086 Microsystem Software Development, as illustrated by the following benefits of PL/M use:

Cost-Effective Alternative to Assembly Language

PL/M 8086 programs are code efficient. PL/M 8086 combines all of the benefits of a high-level language (ease of use, high productivity) with the ability to access the 8086 architecture. Consequently, for the development of systems software, PL/M 8086 is the cost-effective alternative to assembly language programming.

Low Learning Effort

PL/M is easy to learn and to use, even for the novice programmer.

Earlier Project Completion

Critical projects are completed much earlier than otherwise possible because PL/M 8086, a structured high-level language, increases programmer productivity.

Lower Development Cost

Increases in programmer productivity translate immediately into lower software development costs because fewer programming resources are required for a given programmed function.

Increased Reliability

PL/M 8086 is designed to aid in the development of reliable software (PL/M 8086 programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.

Easier Enhancements and Maintenance

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

SPECIFICATIONS

Documentation Package

PL/M-8086 User's Guide for 8086-based Development Systems

SUPPORT:

Hotline Telephone Support, Software Performance Reporting (SPR), Software Updates, Technical Reports, Monthly Newsletter available.

ORDERING INFORMATION

Order Code	Operating Environment
D86PLM86	IBM PC XT/AT running PCDOS Version 3.0 or later
R86PLM86	Intel System 8086/3XX running iRMX™ 86
WSPLM86	VAX/VMS
MVVSPLM86	MICROVAX/VMS

iC-86/186
C COMPILER FOR THE 8086 AND THE 80186

- **Implements Full C Language as Defined by the Draft ANSI Standard**
- **Produces High Density Code Rivaling Assembler**
- **Supports Both Standard Intel (PL/M-like) and Standard C Calling Conventions**
- **Allows Mixed Memory Mode/ Programming via Near and Far Pointers**
- **Available for DOS and VAX/VMS* Operating System**
- **Designed to Work with Intel Debuggers such as I²ICE and PSCOPE**
- **Supports Small, Medium, Compact, and Large Models of Computation**
- **Supports IEEE Floating Point Math with 8087 Coprocessor**
- **Supports I/O and Hardware Interrupts Directly in C**
- **Supports Full Standard I/O Library (STDIO)**
- **Written in C**
- **All Code and Libraries Are Fully Compatible**

The C Programming Language was originally designed in 1972 and has become increasingly popular as a systems development language. C combines the flexibility and programming speed of a higher level language with the efficiency and control of assembly language.

Intel iC-86 brings the full power of the C programming language to 8086, 8088, 80186, and 80188 based microprocessor systems. iC-86 has been developed specifically for embedded microprocessor-based applications.

Intel iC-86 supports the full C language as described in the Kernighan and Ritchie book, "The C Programming Language", (Prentice-Hall, 1978). iC-86 implements the complete C language specification as defined in the ANSI X3J11 standard.

iC-86 is an outstanding microprocessor system implementation language because it provides:

1. the ability to manipulate the fundamental objects of the machine (including machine addresses) as easily as assembly language.
2. the power and speed of a structured language supporting a large number of data types, storage classes, expressions and statements,
3. processor independence (most programs developed for other processors can be easily transported to the 8086), and
4. code that rivals assembly language in efficiency

INTEL iC-86 COMPILER DESCRIPTION

The iC-86 compiler operates in four phases: preprocessor, parser, code generator, and optimizer. The preprocessor phase interprets directives in C source code, including conditional compilations (# define). The parser phase converts the C program into an intermediate free form and does all syntactic and semantic error checking. The code generator phase converts the parser's output into an efficient intermediate binary code, performs constant folding, and features an extremely efficient register allocator, ensuring high quality code. The optimizer phase converts the output of the code gener-

ator into relocatable Intel Object Module Format (OMF) code, without creating an intermediate assembly file. Optionally, the iC-86 compiler can produce a symbolic pseudo-assembly file. The iC-86 optimizer eliminates common code, eliminates redundant loads and stores, and resolves span dependencies (shortens branches) within a program.

The iC-86 runtime library consists of a number of functions which the C programmer can call. The runtime system includes the standard I/O library (STDIO), conversion routines, routines for manipulating strings, special routines to perform functions not available on the 8086 (32-bit arithmetic and emulated floating point), and (where appropriate) routines for interfacing with the operating system.

iC-86 uses Intel's linker and locator and generates debug records for symbols and lines on request, permitting access to Intel's PSCOPE AND I^2ICE™ to aid in program testing. Intel's DOS LINK86 can also be used to create DOS executable .EXE files for prototyping.

FEATURES

Memory Model Support

iC-86 supports the SMALL, MEDIUM, COMPACT, and LARGE segmentation models. A SMALL Model Program can have up to 64K bytes of code space and 64K bytes of total data, memory, and stack space for all combined modules. SMALL model will generate the most efficient code and is the compiler default. A MEDIUM Model Program can have a seperate 64K segment for each module of code, while total data, memory, and stack must be less than 64K. In the COMPACT model code, data, stack, and memory can each reside in a separate 64K segment. The LARGE model is intended for programs needing up to 64K of code space and 64K of data space for each module. LARGE model also provides up to 64K of stack space and up to 64K of space for memory. Mixed model programming is supported with "near" and "far" calls.

Preprocessor Directives

#define—defines a macro

#include—includes code outside of the program source file

#if—conditionally includes or excludes code

Other preprocessor directives include #undef, #ifdef, #ifndef, #else, #endif, and #line.

Statements

The C language supports a variety of statements:

- Conditionals: IF, IF-ELSE
- Loops: WHILE, DO-WHILE, FOR
- Selection of cases: SWITCH, CASE, DEFAULT
- Exit from a function: RETURN
- Loop control: CONTINUE, BREAK
- Branching: GOTO

Expressions and Operators

The C language includes a rich set of expressions and operators.

Primary expression: invoke functions, select elements from arrays, and extract fields from structures or unions

Arithmetic operators: add, subtract, multiply, divide, modulus

Relational operators: greater than, greater than or equal, less than, less than or equal, not equal

Unary operators: indirect through a pointer, compute an address, logical negation, ones complement, provide the size in bytes of an operand.

Logical operators: AND, OR

Bitwise operators: AND, exclusive OR, inclusive OR, bitwise complement

Calling Conventions

iC-86 provides two distinct calling conventions for handling the way parameters are passed on the stack. The *variable parameter list* (VPL) is the default, and is consistent with most other C compilers. VPL pushes the last (rightmost) parameter first, and the first parameter is pushed last. The *fixed parameter list* (FPL) is the calling convention for most other Intel compilers, including PL/M. FPL pushes the first parameter first, and the last parameter last. By using the keyword "alien", the user can make direct PL/M calls.

Data Types and Storage Classes

Data in C is described by its type and storage class. The type determines its representation and use, and the storage class determines its lifetime, scope, and storage allocation. The following data types are fully supported by iC-86.

char

an 8-bit signed integer

int

a 16-bit signed integer

short

same as int (on the 8086)

long

a 32-bit signed integer

unsigned

a modifier for integer data types (char, int, short, and long) which doubles the positive range of values

float

a 32-bit floating point number which utilizes the 8087 or a software floating point library

double

a 64-bit floating point number

bit-field

maximum size is that of an int

void

a special type that cannot be used as an operator; normally used for functions called only for effect (to prevent their use in contexts where a value is required).

enum

an enumerated data type

These fundamental data types may be used to create other data types including: arrays, functions, structures, pointers, and unions.

The storage classes available in iC-86 include:

register

suggests that a variable be kept in a machine register, often enhancing code density and speed

extern

a variable defined outside of the function where it is declared; retaining its value throughout the entire program and accessible to other modules

auto

a local variable, created when a block of code is entered and discarded when the block is exited

static

a local variable that retains its value until the termination of the entire program

typedef

defines a new data type name from existing data types

BENEFITS

Faster Compilation

Intel iC-86 compiles C programs substantially faster than standard C compilers because it produces Intel OMF code directly, eliminating the traditional intermediate process of generating an assembly file.

Portability of Code

Because Intel iC-86 supports the STDIO and produces Intel OMF code, programs developed on a variety of machines can easily be transported to the 8086.

Rapid Program Development

Intel iC-86 provides the programmer with detailed error messages and access to PSCOPE-86 and I²ICE to speed program development. A complete listing file can also be produced.

Full Manipulation of the 8086 and 80186

Intel iC-86 enables the programmer to utilize features of the C language to control bit fields, pointers, addresses and register allocation, taking full advantage of the fundamental concepts of the 8086. A MOD186 control is also available to provide full support for the additional instructions in the 80186.

SPECIFICATIONS

Operating Environment

The iC-86 compiler runs host resident on DOS 3.0 or greater. iC-86 can also run as a cross compiler on a VAX 11/780 computer under the VMS operating system. 640K bytes of User Memory is required on all versions. The PC DOS Operating Environment is also supported. Specify desired version when ordering.

Required Hardware

VAX version:

— Digital Equipment Corporation VAX 11/780 or compatible computer running VMS 4.5 or greater

PC DOS version:

— PC XT or AT using PC DOS 3.0 or later

— Hard disk recommended

Required Software

MicroVAX or VAX version:

— VMS Operating System 4.5 or greater

PC DOS version:

— PC DOS Release 3.0 or later Operating System

Documentation Package

iC-86 User Manual

C: A Reference Manual by Harbison and Steele (1987 Prentice-Hall)

Shipping Media

VAX version:

— 1600 bpi, 9 track Magnetic tape

DOS version:

— 5¼" DOS format diskette

— 3.5" DOS format diskette

ORDERING INFORMATION

Order Code	Description
MVVSC86	iC-86 Cross Compiler for MicroVAX/VMS
VVSC86	iC-86 Cross Compiler for VAX/VMS
D86C86	iC-86 Compiler for PC DOS
Intel Software License required for VAX and MicroVAX versions	

SUPPORT

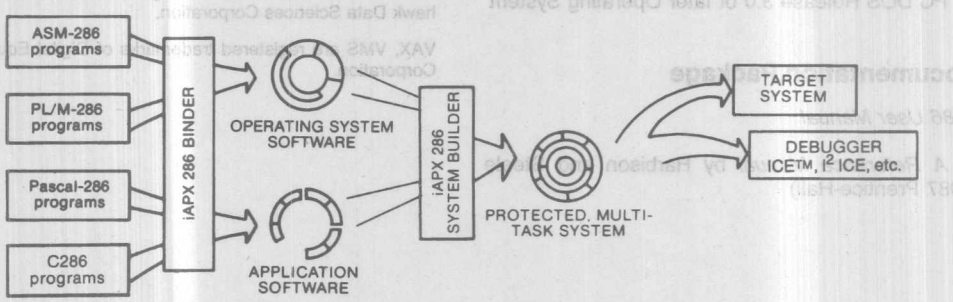
Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

VAX, VMS are registered trademarks of Digital Equipment Corporation.

286 SOFTWARE DEVELOPMENT TOOLS AVAILABLE ON CHOICE OF INDUSTRY STANDARD HOSTS INCLUDING PC-DOS AND VAX/VMS*

- **286 Software Development Package**
  - **— Complete System Development Capability for High-Performance 286 Applications**
  - **— Allows Creation of Multi-User, Virtual Memory, and Memory-Protected Systems**
  - **— Macro Assembler for Machine-Level Programming**
- **Pascal-286 Software Package**
  - **— High-Level Programming Language for the Protected Virtual Mode of the 286**
  - **— Implements ISO Standard Pascal**
- **PL/M 286 Software Package**
  - **— Systems Programming Language for the Protected Virtual Address Mode of the 286**
  - **— Advanced Structured System Implementation Language for Algorithm Development**
- **IC-286, C Compiler for the 80286**
  - **— Implements Full C Language**
  - **— High-Level Language for the Protected Virtual Mode of the 286**



231665-1

The iAPX 286 Software Development Package keeps the protection mechanism invisible to the application programmer, yet easy to configure for the system programmer.

*VAX/VMS are trademarks of Digital Equipment Corporation.

September 1987
Order Number: 231665-003

80286 MACRO ASSEMBLER

- **Instruction Set and Assembler Mnemonics Are Upward Compatible with ASM-86/88**
- **Powerful and Flexible Text Macro Facility**
- **Type-Checking at Assembly Time Helps Reduce Errors at Run-Time**
- **Structures and RECORDS Provide Powerful Data Representation**
- **"High-Level" Assembler Mnemonics Simplify the Language**
- **Supports Full Instruction Set of the 80286/20, Including Memory Protection and Numerics**

ASM-286 is the "high-level" macro assembler for the 80286 assembly language. ASM-286 translates symbolic assembly language mnemonics into relocatable object code. The assembler mnemonics are a superset of ASM-86/88 mnemonics; new ones have also been added to support the new 80286 instructions. The segmentation directives have been greatly simplified.

The 80286 assembly language includes approximately 150 instruction mnemonics. From these few mnemonics the assembler can generate over 4,000 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 150 mnemonics to generate all possible machine instructions. ASM-286 will generate the shortest machine instruction possible (given explicit information as to the characteristics of any forward referenced symbols).

The powerful macro facility in ASM-286 saves development and maintenance time by coding common program sequences only once. A macro substitution is made each time the sequence is to be used. This facility also allows for conditional assembly of certain program sequences.

ASM-286 offers many features normally found only in high-level languages. The assembly language is strongly typed, which means it performs extensive checks on the usage of variables and labels. This allows many programming errors to be detected when the program is asembled, long before it is being debugged.

ASM-286 object modules conform to a thorough, well-defined format used by all 286 high-level languages and utilities. This makes it easy to call (and be called from) HLL object modules.

Key Benefit

For programmers who wish to use assembly language. ASM-286 provides many powerful "high-level" capabilities that simplify program development and maintenance.

80286 BINDER

- **Links Separately Compiled Program Modules Into an Executable Task**
- **Makes the 80286 Protection Mechanism Invisible to Application Programmers**
- **Works with PL/M-286, Pascal-286, FORTRAN-286, ASM-286 Object Modules and IC-286**
- **Performs Incremental Linking with Output of Binder and Builder**
- **Resolves PUBLIC/EXTERNAL Code and Data References, and Performs Intermodule Type-Checking**
- **Provides Print File Showing Segment Map, Errors and Warnings**
- **Assigns Virtual Addresses to Tasks in the 2^{32} Address Space**
- **Generates Linkable or Loadable Module for Debugging**

BND-286 is a utility that combines 80286 object modules into executable tasks. In creating a task, the Binder resolves Public and External symbol references, combines segments, and performs address fix-ups on symbolic code and data.

The Binder takes object modules written in ASM-286, PL/M-286, Pascal-286, FORTRAN-286 or iC-286 and generates a loadable module (for execution or debugging), or a linkable module (to be re-input to the Binder later; this is called incremental binding). The binder accepts library modules as well, linking only those modules required to resolve external references. BND-286 generates a print file displaying a segment map, and error messages.

The Binder is used by system programmers and application programmers. Since application programmers need to develop software independent of any system architecture, the 286 memory protection mechanism is "hidden" from users of the Binder. This allows application tasks to be fully debugged before becoming part of a protected system. (A protected system may be debugged as well.) System protection features are specified later in the development cycle, using the 286 System Builder. It is possible to link operating system services required by a task using either the Binder or the Builder. This flexibility adds to the ease of use of the 286 utilities.

Key Benefits

The Binder is the only utility an application programmer needs to develop and debug an individual task. Users of the Binder need not be concerned with the architecture of the target machine, making application program development for the 286 very simple.

80286 MAPPER

- **Flexible Utility to Display Object File Information**
- **MAP-286 Selectively Purges Symbols from a Load Module**
- **Provides Inter-Module Cross-Referencing for Modules Written in All Languages**
- **Mapper Allows Users to Display:**
  - **Protection Information:**
    - **Segment Tables**
    - **Gate Tables**
    - **Public Addresses**
  - **Debug Information:**
    - **Module Names**
    - **Program Symbols**
    - **Line Numbers**

Key Benefit

A cross-reference map showing references *between* modules simplifies debugging; the map also lists and controls all symbolic information in one easy-to-read place.

80286 LIBRARIAN

- **Fast, Easy Management of 80286 Object Module Libraries**
- **Only Required Modules Are Linked, When Using the Binder or Builder**
- **Librarian Allows User to:**
  Create Libraries
  Add Modules
  Replace Modules
  Delete Modules
  Copy Modules from Another Library
  Save Library Module to Object File
  Create Backup
  Display Module Information (Creation Date, Public, Segments)

Key Benefit

Program libraries improve management of program modules, and reduce software administrative overhead.

80286 SYSTEM BUILDER

- **Supports Complete Creation of Protected, Multi-Task Systems**
- **Resolves PUBLIC/EXTERNAL Definitions (Between Protection Levels)**
- **Supports Memory Protection by Building System Tables, Initializing Tasks, and Assigning Protection Rights to Segments**
- **Creates a Memory Image of a 286 System for Cold-Start Execution**
- **Target System may be Boot-Loadable, Programmed into ROM, or Loaded From Mass-Store**
- **Generates Print File with Command Listing and System Map**

BLD-286 is the utility that lets system programmers configure mutli-tasking, protected systems from an operating system and discrete tasks. The Builder generates a cold-start execution module, suitable for ROM-based or disk-based systems.

The Builder accepts input modules from 80286 translators or the 80286 Binder. It also accepts a "Build File" containing definitions and initial values for the 286 protection mechanism—descriptor tables, gates, segments, and tasks. BLD-286 generates a Loadable or bootloadable output module, as well as a print file with a detailed map of the memory-protected system.

Using the Builder command Language, system programmers may perfrom the following functions:

— Assign physical addresses to segments; also set segment access rights and limits.
— Create Call, Trap, and Interrupt "Gates" (entry-points) for inter-level program transfers.
— Make gates available to tasks; this is an easier way to define program interfaces than using interface libraries.
— Create Global (GDT), Interrupt (IDT), and any Local (LDT) Descriptor Tables.
— Create Task State Segments and Task Gates for multi-task applications.
— Resolve inter-module and inter-level references, and perform type-checking.
— Automatically select required modules from libraries.
— Configure the memory image into partitions in the address space.
— Selectively generate an object file and various sections of the print file.

Key Benefit

Allows a system programmer to define the configuration of a protected system in *one* place, with one easy-to-use Utility. This specification may then be adopted by all project members, using either the Builder *or just the Binder*. The flexibility simplifies program development for all users.

SPECIFICATIONS

Documentation

ASM 286 Language Reference Manual
ASM 286 Macro Assembler Operating Instructions
80286 Utilities User's Guide
80286 System Builder User's Guide
Pocket Reference for all the above:
ASM 286
Utilties

SUPPORT AVAILABLE

Hotline Telephone Support, Software Updates, Technical Reports

ORDERING INFORMATION

Product Code	Operating Environment
I86 ASM 286	Series III/Series IV
D86 ASM 286	IBM PC XT/AT running PCDOS 3.0 or later
VVS ASM 286	VAX/VMS
MVVS ASM 286	MicroVAX/VMS
X286 ASM 286	Xenix for Intel 286/3XX Systems
R286 ASM 286	RMX 286 for Intel 286/3XX Systems

PASCAL-286 SOFTWARE PACKAGE

- **High-Level Programming Language for the Protected Virtual Mode iAPX 286**
- **Implements ISO Standard Pascal Many Useful Extensions may be Enabled via a Compiler Switch**
- **Choice of Industry Standard Hosts**
- **Supports Full Symbolic Debugging with iAPX 286 Software and ICE™ Debuggers**
- **Upward Compatible with Pascal-86 for Software Portability**
- **Produces Relocatable Object Code Which is Linkable to Object Modules Generated by Other iAPX 286 Translators**
- **Fully Supports the 80287 Numeric Processor using the IEEE Floating Point Standard**

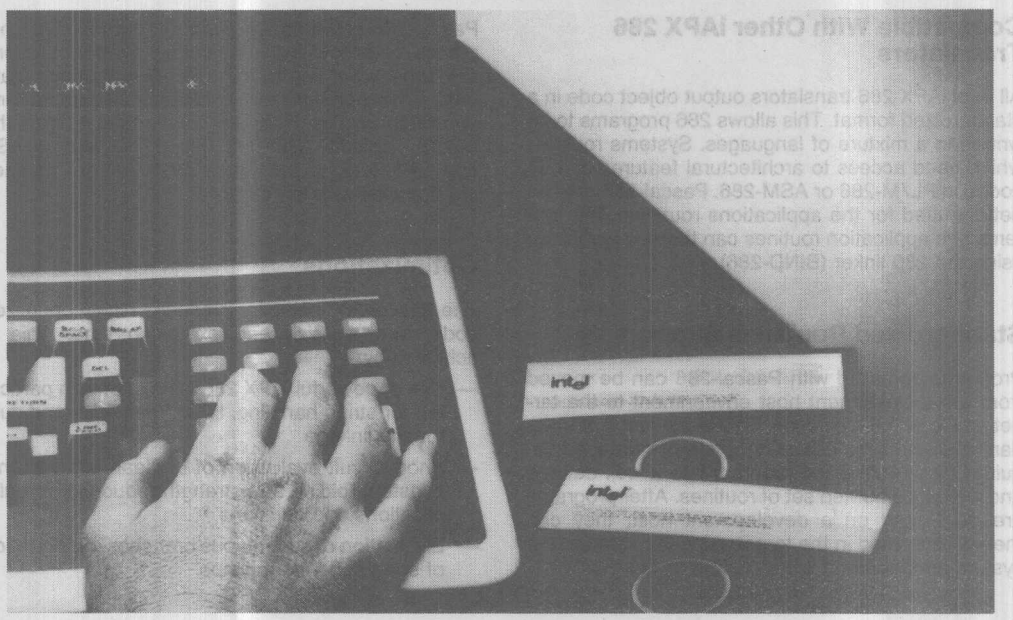
Pascal-286 is a powerful, structured, applications programming language for the protected virtual address mode of the iAPX 286. Pascal-286 is upward compatible with Pascal-86 so that 8086 Pascal source code can be ported to the iAPX 286 in protected mode.

Pascal-286 implements strict ISO standard Pascal, but with many useful extensions. These include separate compilation of modules, interrupt handling, port I/O, and 80287 numerics support. A control is provided in the compiler to flag all non-ISO features used.

Pascal-286 produces relocatable object code which can be linked with object code produced by other iAPX 286 translators such as ASM-286 and PL/M-286. Thus, a combination of translators can be used to provide great programming flexibility.

Type and symbol information needed by software and in-circuit debuggers is added to the object code by the Pascal-286 compiler. This information can be stripped off by the compiler or linker for the final production version.

The Pascal-286 compiler runs on the Intel Microcomputer Development Systems (Series III/Series IV) as well as the IBM PC XT/AT running PCDOS version 3.0 or later.



230863-1

November 1986
Order Number: 230863-001

FEATURES

Conforms to ISO Standard Pascal

Pascal has gained wide acceptance as a portable language for microcomputer applications. However, portability can result only if standards are adhered to. Pascal-286 is a strict implementation of ISO standard Pascal. Extensions are provided to make the language more powerful for microprocessor applications. All extensions are clearly highlighted in the documentation. In addition, the compiler provides a control to flag any non ISO feature used. Pascal-286 will evolve to track future enhancements to standard Pascal.

Upward Compatible with Pascal-86

The Pascal-286 compiler produces object code for the protected virtual address mode of the iAPX 286 language. However, no 286 architecture specific features have been added to the Pascal-286 language. This makes Pascal-286 source code upward compatible with Pascal-86, which allows for porting of 8086 software to the protected 286 with relative ease.

Compatible With Other iAPX 286 Translators

All Intel iAPX 286 translators output object code in a standardized format. This allows 286 programs to be written in a mixture of languages. Systems routines which need access to architectural features can be coded in PL/M-286 or ASM-286. Pascal-286 may be better suited for the applications routines. The systems and application routines can then be combined using the 286 linker (BIND-286).

Standardized Run Time Support

Programs compiled with Pascal-286 can be moved from the development host environment to the target environment with ease. This is the result of standardizing run-time operating system interfaces required by the compiled program into a well defined and well documented set of routines. After programs are developed on a development host, they can then be executed in the target using the same set of system interfaces.

Extensions for Microprocessor Programming

Pascal-286 provides extensions that make it powerful for microprocessor applications. Built-in procedures allow I/O directly from the ports of the iAPX 286. This speeds up I/O as it is done by direct communication with the microprocessor. Interrupt processing is also supported by built in procedures. Examples are: ENABLEINTERRUPTS, DISABLEINTERRUPTS, CAUSEINTERRUPT. Many built in procedures and variables are provided for communicating with the 80287 for numeric computations.

Compiler Controls

The Pascal-286 compiler provides many controls which can be used at invocation time to enhance programming flexibility. Examples are: CODE/NOCODE, DEBUG/NODEBUG, INCLUDE (file), LIST/NOLIST, OPTIMIZE (n), EXTENSIONS/NOEXTENSIONS. All controls have default values that are active unless the opposite is specified during invocation. Thus, for most compiles, no controls need be specified.

Support for IEEE Standard Numerics

Pascal-286 provides full support for the 80287 numerics co-processor. All floating point operations are done according to the IEEE floating point standard. The benefits are predictable, accurate and consistent results. Built-in procedures to support the 80287 include GET8087ERRORS and MASK 8087ERRORS. A full set of 80287 library routines are supplied with the compiler.

Optimizations

The Pascal-286 compiler produces highly optimized code, both in size and execution time. This is achieved by:

- Use of powerful iAPX 286 instructions, in particular, for string handling, 80287 numerics and subroutine linkage
- Short circuit evaluation of boolean expressions, constant folding and strength reduction of multiplications and additions
- Elimination of superfluous branches, optimization of span dependent jumps

Support Available

Hotline service, Software Updates and technical newsletters.

ORDERING INFORMATION

Product Code	Operating Environment	Documentation Package
I86 PAS 286	Intel Series III/Series IV	Pascal-286 User's Guide
D86 PAS 286	IBM PC XT/AT running PCDOS version 3.0 or later	Pascal-286 Pocket Reference

PL/M 286 SOFTWARE PACKAGE

- **System Programming Language for the Protected Virtual Address Mode 80286**
- **Upward Compatible with PL/M 86 and PL/M 80 Assuring Software Portability**
- **Enhanced to Support Design of Protected, Multi-User, Multi-Tasking, Virtual Memory Operating System Software**
- **Multiple Levels of Optimization**
- **Advanced, Structured System Implementation Language for Algorithm Development**
- **Produces Relocatable Object Code Which is Linkable to Object Modules Generated by all Other 80286 Language Translators**
- **Wide Choice of Industry Standard Hosts**

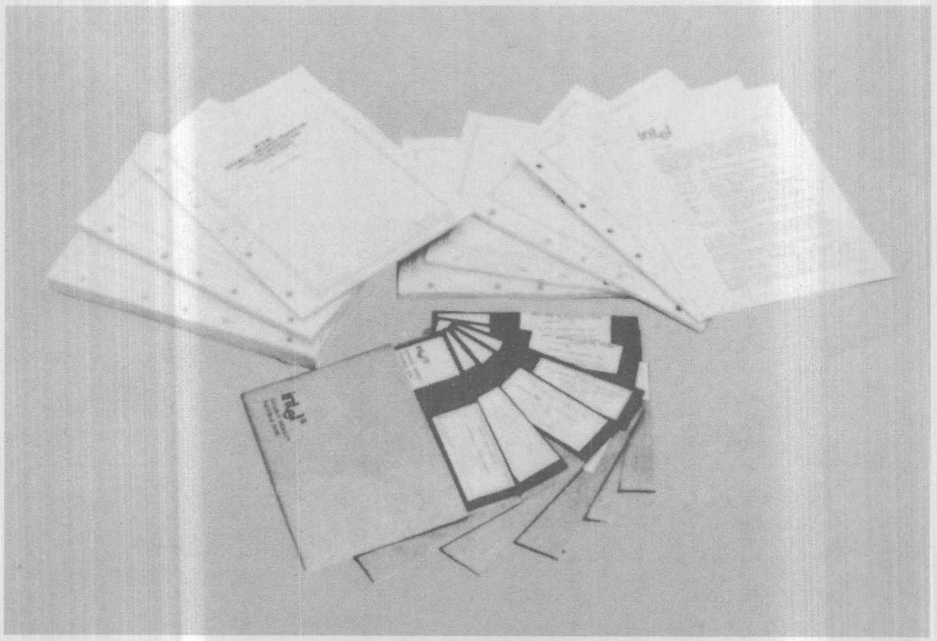
PL/M 286 is a powerful, structured, high-level system implementation language for the development of system software for the protected virtual address mode 80286. PL/M 286 has been enhanced to utilize 80286 features—memory management and protection—for the implementation of multi-user, multi-tasking virtual memory operating systems.

PL/M 286 is upward compatible with PL/M 86 and PL/M 80. Existing systems software can be recompiled with PL/M 286 to execute in protected virtual address mode on the 80286.

PL/M 286 is the high-level alternative to assembly language programming on the 80286. For the majority of 80286 systems programs, PL/M 286 provides the features needed to access and to control efficiently the underlying 80286 hardware and consequently it is the cost-effective approach to develop reliable, maintainable system software.

The PL/M 286 compiler has been designed to efficiently support all phases of software development features such as a built-in syntax checker, multiple levels of optimization, virtual symbol table and four models of program size and memory usage for efficient code generation provide the total program development support needed.

The PL/M 286 compiler runs on the Intel Microcomputer Development Systems (Series III/Series IV) as well as the IBM PC XT/AT running PC DOS version 3.0 or later, Digital Equipment VAX/VMS† Systems, and Intel XENIX** 286 and RMX 286 based systems.



280335-1

†VAX, VMS are trademarks of Digital Equipment Corporation.
**XENIX is a trademark of Microsoft Corporation.

October 1987
Order Number: 280335-002

FEATURES

Major features of the Intel PL/M 286 compiler and programming language include:

Structured Programming

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible by clearly defining the scope of user variables (local to a private procedure, for example).

The use of modules and procedures to break down a large problem leads to productive software development. The PL/M 286 implementation of block structure allows the use of REENTRANT procedures, which are especially useful in system design.

Language Compatibility

PL/M 286 object modules are compatible with object modules generated by all other 286 translators. This means that PL/M programs may be linked to programs written in any other 286 language.

Object modules are compatible with In-Circuit Emulators; DEBUG compiler control provides the In-Circuit Emulators with full symbolic debugging capabilities.

PL/M 286 language is upward compatible with PL/M 86 and PL/M 80 so that application programs may be easily ported to run on the protected mode 80286.

Supports Seven Data Types

PL/M makes use of seven data types for various applications. These data types range from one to four bytes and facilitate various arithmetic, logic, and addressing functions:

- Byte: 8-bit unsigned number
- Word: 16-bit unsigned number
- Dword: 32-bit unsigned number
- Integer: 16-bit signed number
- Real: 32-bit floating-point number
- Pointer: 16-bit or 32-bit memory address indicator
- Selector: 16-bit pointer base

Another powerful facility allows the use of BASED variable which permit run-time mapping of variables to memory locations. This is especially useful for passing parameters, relative and absolute addressing, and dynamic memory allocation.

Two Data Structuring Facilities

In addition to the seven data types and based variables, PL/M supports two powerful data structuring facilities. These help the user to organize data into logical groups.

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of both: Arrays of structures or structures of arrays

Numerics Support

PL/M programs that use 32-bit REAL data are executed using the 80287 Numeric Data Processor for high performance. All floating-point operations supported by PL/M are executed on the 80287 according to the IEEE floating-point standard. PL/M 286 programs can use built-in functions and predefined procedures—INIT$REAL$MATH$UNIT, SET$REAL$MODE, GET$REAL$ERROR, SAVE$REAL$STATUS, RESTORE$REAL$STATUS,—to control the operation of the 80287 within the scope of the language.

Built-In String Handling Facilities

The PL/M 286 language contains built-in functions for string manipulation. These byte and word functions perform the following operations on character strings: MOVE, COMPARE, TRANSLATE, SEARCH, SKIP, and SET.

Built-In Port I/O

PL/M 286 directly supports input and output from the 80286 ports for single BYTE and WORD transfers. For BLOCK transfers, PL/M 286 programs can make calls to predefined procedures.

Interrupt Handling

PL/M 286 has the facility for generating and handling interrupts on the 80286. A procedure may be defined as an interrupt handler through use of the INTERRUPT attribute. The compiler will then generate code to save and restore the processor status on each execution of the user-defined interrupt han-

dler routine. The PL/M statement CAUSE$INTERRUPT allows the user to trigger a software interrupt from within the program.

Protection Model

PL/M 286 supports the implementation of protected operating system software by providing built-in procedures and variables to access the protection mechanism of the 80286. Predefined variables—TASK$REGISTER, LOCAL$TABLE, MACHINE$STATUS, etc.—allow direct access and modification of the protection system. Untyped procedures and functions—SAVE$GLOBAL$TABLE, RESTORE$GLOBAL$TABLE, SAVE$INTERRUPT$TABLE, RESTORE$INTERRUPT$TABLE, CLEAR$TASK$SWITCHED$FLAG, GET$ACCESS$RIGHTS, GET$SEGMENT$LIMIT, SEGMENT$READABLE, SEGMENT$WRITEABLE, ADJUST$RPL—provide all the facilities needed to implement efficient operating system software.

Compiler Controls

The PL/M 286 compiler offers controls that facilitate such features as:

- Optimization
- Conditional compilation
- The inclusion of additional PL/M source files from disk
- Cross-reference of symbols
- Optional assembly language code in the listing file
- The setting of overflow conditions for run-time handling

Addressing Control

The PL/M 286 compiler uses the SMALL, COMPACT, MEDIUM, and LARGE control to generate optimum addressing instructions for programs. Programs of any size can be easily modularized into "subsystems" to exploit the most efficient memory addressing schemes. This lowers total memory requirements and improves run-time execution of programs.

Code Optimization

The PL/M 286 compiler offers four levels of optimization for significantly reducing overall program size.

- Combination or "folding" of constant expressions; and short-circuit evaluation of Boolean expressions
- "Strength reductions": a shift left rather than multiply by 2; and elimination of common sub-expressions within the same block
- Machine code optimizations; elimination of superfluous branches; reuse of duplicate code; removal of unreachable code
- Optimization of based-variable operations and cross-statement load/store

Error Checking

The PL/M 286 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed and helpful set of programming and compilation error messages is provided by the compiler and user's guide.

BENEFITS

PL/M 286 is designed to be an efficient, cost-effective solution to the special requirements of protected mode 80286 Microsystem Software Development, as illustrated by the following benefits of PL/M use:

Low Learning Effort

PL/M 286 is easy to learn and use, even for the novice programmer.

Earlier Project Completion

Critical projects are completed much earlier than otherwise possible because PL/M 286, a structured high-level language, increases programmer productivity.

Lower Development Cost

Increases in programmer productivity translate immediately into lower software development costs because less programming resources are required for a given programmed function.

Increased Reliability

PL/M 286 is designed to aid in the development of reliable software (PL/M 286 programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.

Easier Enhancements and Maintenance

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

Cost-Effective Alternative to Assembly Language

PL/M 286 programs are code efficient. PL/M 286 combines all of the benefits of a high-level language (ease of use, high productivity) with the ability to access the 80286 architecture. This includes language features for control of the 80286 protection mechanism. Consequently, for the development of systems software, PL/M 286 is the cost-effective alternative to assembly language programming.

Support Available

90 Days:
Hotline Telephone Support, Software Updates, Subscription Service

Documentation Package

PL/M 286 User's Guide
PL/M 286 Pocket Reference

ORDERING INFORMATION

Ordering Code	Operating Environment
D86PLM286	IBM PC XT/AT running PCDOS version 3.0 or later
VVSASM286	VAX, VMS
X286PLM286	Xenix for Intel Systems 286/3XX
R286PLM286	iRMX™ 286 for Intel Systems 286/3XX

386 SOFTWARE TOOLS

PL/M 386 Software Package

- Systems Programming Language for the Protected Virtual Address Mode 386
- Upward Compatible with PL/M 286, PL/M 86, and PL/M 80 Assuring Software Portability

386 Relocation, Linkage and Library Tools

- Provides System Development Capability for High-Performance 386 Applications
- Allows Creation of Multi-User Virtual Memory, and Memory-Protected Systems

C 386

- Implements Full C Language and New Extensions
- Produces High Density Code Rivaling Assembler
- Supports Intel Object Module Format (OMF)

ASM 386

- Instruction Set and Assembler Mnemonics Are Upward Compatible with ASM 286 and ASM 86
- Type-Checking at Assembly Time Helps Reduce Errors at Run-Time

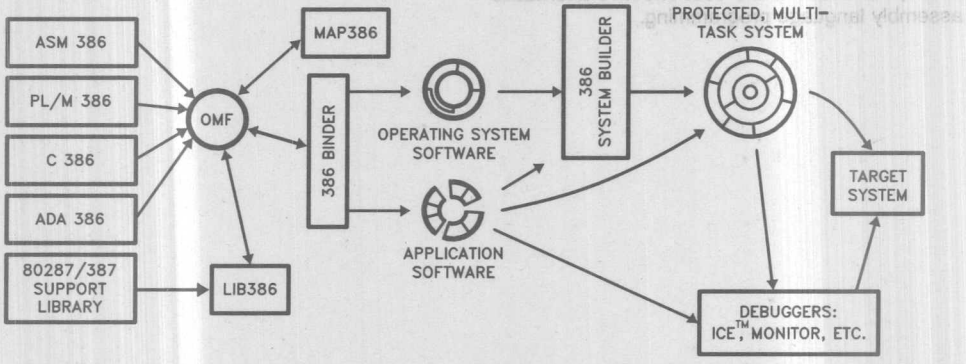


Figure 1. Development Environment Tools for the 386

386 Software tools are available on industry standard hosts, including VAX/VMS, PC-DOS, and XENIX*

September 1987
Order Number: 231637-003

ASM 386

- **Instruction Set and Assembler Mnemonics Are Upward Compatible with ASM 286 and ASM 86**
- **Powerful and Flexible Text Macro Facility**
- **Type-Checking at Assembly Time Helps Reduce Errors at Run-Time**
- **Structures and Records Provide Powerful Data Representation**
- **"High-Level" Assembler Mnemonics Simplify the Language**
- **Supports Full Instruction Set of the 386, Including Memory Protection and Numerics**
- **Supports 286 Addressing Modes**
- **DOS Version Supports Expanded Symbol Table if Used with Intel Above Board**

ASM 386 is the "high-level" macro assembler for the 386 assembly language. ASM 386 translates symbolic assembly language mnemonics into relocatable object code. The assembler mnemonics are a superset of ASM 286/86/88 mnemonics; new ones have also been added to support the new 386 instructions. The segmentation directives have been greatly simplified.

The 386 assembly language includes approximately 275 instruction mnemonics. From these few mnemonics the assembler can generate over 40,000 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 275 mnemonics to generate all possible machine instructions. ASM 386 will generate the shortest machine instruction possible (given explicit information as to the characteristics of any forward referenced symbols).

The powerful macro facility in ASM 386 saves development and maintenance time by coding common program sequences only once. A macro substitution is made each time the sequence is to be used. This facility also allows for conditional assembly of certain program sequences.

ASM 386 offers many features normally found only in high-level languages. The assembly language is strongly typed, which means it performs extensive checks on the usage of variables and labels. This means that many programming errors will be detected when the program is assembled, long before it is being debugged.

ASM 386 object modules conform to a thorough, well-defined format used by all 386 high-level languages and utilities. This means it is easy to call (and be called from) High Level Language object modules.

SUPPORT

Hotline Telephone Support, Software Performance Report (SPR), Software Update, Technical Reports, and Monthly Technical Newsletters are available.

ORDERING INFORMATION

Part Number	Description	Operating Environment
X286ASM386	386 Assembler	286/310 XENIX* System
D86ASM386	386 Assembler	PC-DOS 3.0 or greater
VVSASM386	386 Assembler	VAX/VMS 4.4 or greater
MVVSASM386	386 Assembler	MicroVAX/VMS 4.4 or greater

Documentation Package

ASM 386 Assembly Language Reference Manual
ASM 386 Macro Assembler Operating Instructions

*XENIX™ is a trademark of Microsoft.

386 RELOCATION, LINKAGE AND LIBRARY TOOLS

- **System Development Capability for High-Performance 386 Applications**
- **Allows creation of Multi-User, Virtual Memory, and Memory-Protected Systems**
- **System Utilities for Program Linkage and System Building**
- **Package Supports Program Development with ASM 386, PL/M 386, C 386, Ada 386 and FORTRAN 386.**

The 80386 is a 32-bit microprocessor system with 32-bit addressing, integrated memory protection, and instruction pipelining for high performance. The 386 Relocation, Linkage, and Library Tools are a cohesive set of software design aids for programming the 386 microprocessor system. The package enables system programmers to design protected, multi-user and multi-tasking operating system software, and enables application programmers to develop tasks to run on a protected operating system.

The 386 Relocation, Linkage and Library tools include a program binder (for linking separately compiled modules together), a system builder (for configuring protected multiple-task systems), a cross reference mapper, a program librarian, and the 287/387 support library.

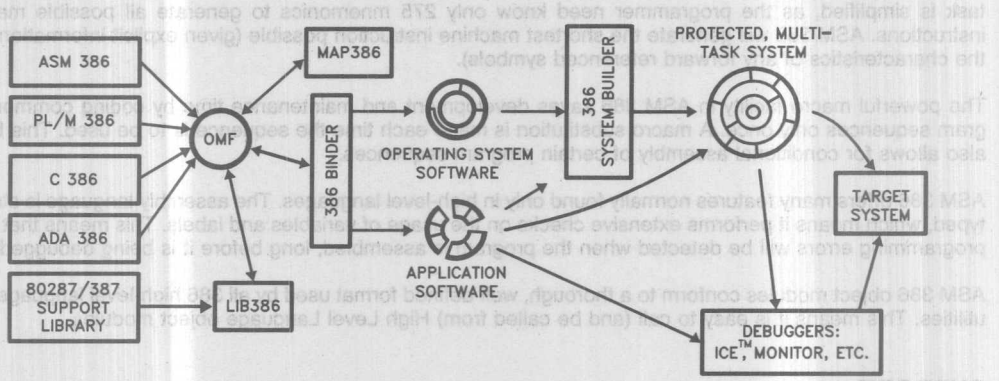


Figure 1. Development Environment Tools for the 386

386 SYSTEM BUILDER

- **Supports Complete Creation of Protected, Multi-task Systems**
- **Resolves PUBLIC/EXTERNAL Definitions (between protection levels)**
- **Supports Memory Protection by Building System Tables, Initializing Tasks, and Assigning Protection Rights to Segments**
- **Creates a Memory Image of a 386 System for Cold-start Execution**
- **Target System may be Boot-loadable, Programmed into ROM, or loaded from Mass-store.**
- **Generates Print File with Command Listing and System Map**

BLD 386 is the utility that lets system programmers configure multi-tasking, protected systems from an operating system and discrete tasks. The Builder generates a cold-start execution module, suitable for ROM-based or disk-based systems.

The Builder accepts input modules from 386 translators or the 386 Binder. It also accepts a "Build File" containing definitions and initial values for the 386 protection mechanism - descriptor tables, gates, segments, and tasks. BLD 386 generates a Loadable or bootloadable output module, as well as a print file with a detailed map of the memory-protected system.

Using the Builder command Language, system programmers may perform the following functions:

— Assign physical addresses to segments; also set segment access rights and limits.

— Create Call, Trap, and Interrupt "Gates" (entry-points) for inter-level program transfers.

— Make gates available to tasks; this is an easier way to define program interfaces than using interface libraries.

— Support Page tables for boot files.

— Create Global (GDT), Interrupt (IDT), and any Local (LDT) Descriptor Tables.

— Create Task State Segments and Task Gates for multi-task applications.

— Resolve inter-module and inter-level references, and perform type-checking.

— Automatically select required modules from libraries.

— Configure the memory image into partitions in the address space.

— Selectively generate an object file and various sections of the print file.

386 BINDER

- **Links Separately Compiled Program Modules Into an Executable Task**
- **Makes the 386 Protection Mechanism Invisible to Application Programmers**
- **Works with PL/M 386, C 386 and ASM 386 Object Modules**
- **Performs Incremental Linking with Output of Binder and Builder**
- **Accepts 386 OMF and 286 OMF to Preserve Software Investment**
- **Resolves PUBLIC/EXTERNAL Code and Data References, and Performs Intermodule Type-Checking**
- **Provides Print File Showing Segment Map, Errors and Warnings**
- **Assigns Virtual Addresses to Tasks in the 2^{32} Address Space**
- **Generates Linkable or Loadable Module for Debugging**

The Binder is the only utility an application programmer needs to develop and debug an individual task. Users of the Binder need not be concerned with the architecture of the target machine, making application program development for the 386 very simple.

BND 386 combines 386 object modules into executable tasks. In creating a task, the Binder resolves Public and External symbol references, combines segments, and performs address fix-ups on symbolic code and data. BND 386 can also combine 286 object modules to preserve software investment.

The Binder takes object modules written in ASM 386, PL/M 386, C 386 and FORTRAN 386 and generates a loadable module (for execution or debugging), or a linkable module (to be re-input to the Binder later; this is called incremental binding). The binder accepts library modules as well, linking only those modules required to resolve external references. BND 386 generates a print file displaying a segment map and error messages.

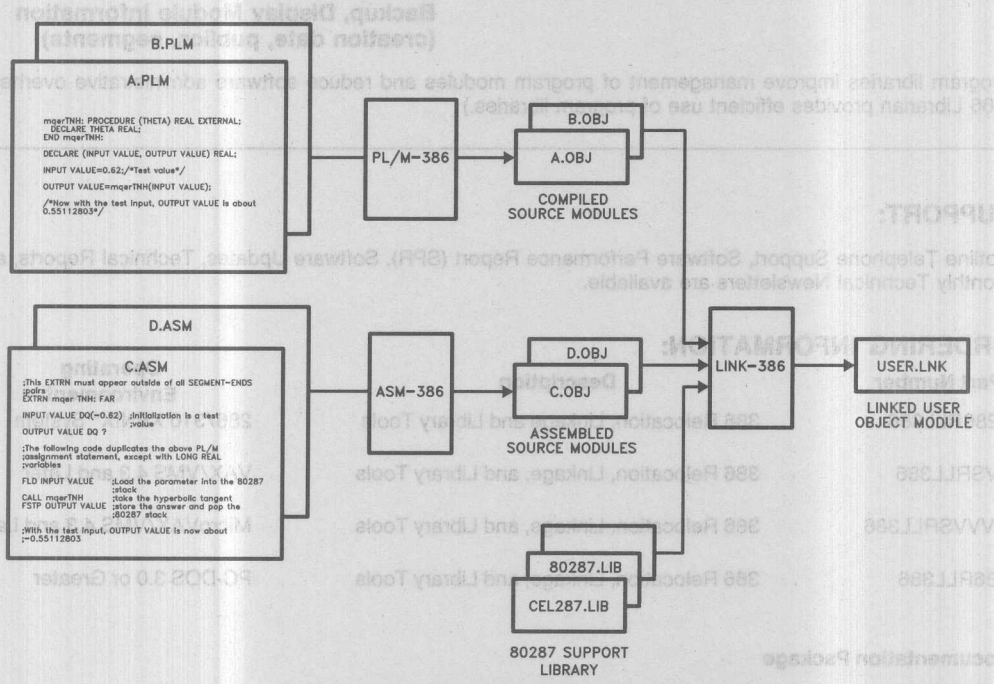
The Binder will be used by system programmers and application programmers. Since application programmers need to develop software independent of any system architecture, the 386 memory protection mechanism is "hidden" from users of the Binder. This allows application tasks to be fully debugged before becoming part of a protected system. (A protected system may be debugged, as well.) System protection features are specified later in the development cycle, using the 386 System Builder. It is possible to link operating system services required by a task using either the Binder or the Builder. This flexibility adds to the ease of the 386 utilities.

80287 SUPPORT LIBRARY

- **Library to support floating point arithmetic in C 386, PL/M 386, ADA 386, ASM 386, and FORTRAN 386**
- **Decimal conversion module supports binary-decimal conversions**
- **Supports proposed IEEE Floating Point Standard for high accuracy and software portability**
- **Common elementary function library provides trigonometric, logarithmic and other useful functions**
- **Error-handler module simplifies floating point error recovery**

The 80287 Support Library provides C 386, PL/M 386, ADA 386, ASM 386 and FORTRAN 386 users with numeric data processing capability. With the Library, it is easy for programs to do floating point arithmetic. Programs can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs. Figure 1 below illustrates how the 80287 Support Library can be bound with PL/M 386 and ASM 386 user code to do this. The 80287 Support Library supports the proposed IEEE Floating Point Standard. Consequently, by using this Library, the user not only saves software development time, but is guaranteed that the numeric software meets industry standards and is portable—the software investment is maintained.

The 80287 Support Library consists of the common elementary function library (CEL287.LIB), the decimal conversion library (DC287.LIB), the error handler module (EH287.LIB) and interface libraries (80287.LIB), (NUL287.LIB).



231637-3

Figure 2. Use of 80287 Support Library with PL/M 386 and ASM 386.

386 MAPPER

- **Flexible Utility to Display Object File Information**
- **MAP 386 Selectively Purges Symbols from a Load Module**
- **Provides Inter-Module Cross-Referencing for Modules Written in All Languages**
- **Supports OS Information**
- **Mapper Allows Users to Display:**
  - **Protection Information**
  - **SEGMENT TABLES**
  - **GATE TABLES**
  - **PUBLIC ADDRESSES**
  - **Debug Information**
  - **MODULE NAMES**
  - **PROGRAM SYMBOLS**
  - **LINE NUMBERS**

The cross-reference map shows references between modules, simplifying debugging. The map also lists and controls all symbolic information in one easy-to-read place.

386 LIBRARIAN

- **Fast, Easy Management of 386 Object Module Libraries**
- **Only Required Modules Are Linked, When Using the Binder or Builder**
- **Librarian Allows Users to: Create Libraries, Add Modules, Replace Modules, Delete Modules, Copy Modules from Another Library, Save Library Module to Object File, Create Backup, Display Module Information (creation date, publics, segments)**

Program libraries improve management of program modules and reduce software administrative overhead. (386 Librarian provides efficient use of program libraries.)

SUPPORT:

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

ORDERING INFORMATION:

Part Number	Description	Operating Environment
X286RLL386	386 Relocation, Linkage and Library Tools	286/310 XENIX* System
VVSRLL386	386 Relocation, Linkage, and Library Tools	VAX/VMS 4.3 and Later
MVVVSRLL386	386 Relocation, Linkage, and Library Tools	MicroVAX/VMS 4.3 and Later
D86RLL386	386 Relocation, Linkage, and Library Tools	PC-DOS 3.0 or Greater

Documentation Package

386 Utilities User's Guide for Xenix* 286 System
386 System Builder User's Guide for Xenix* 286 System
80287 Support Library Reference Manual

*XENIX is a trademark of Microsoft.

PL/M 386 SOFTWARE PACKAGE

- **Systems programming language for the protected virtual address mode 386**
- **Upward compatible with PL/M 286, PL/M 86 assuring software portability**
- **Enchanced to support design of protected, multi-user, multi-tasking, virtual memory operating system software**
- **Produces relocatable object code which is linkable to object modules generated by all other 386 language translators**
- **Advanced, structured system implementation language for algorithm development**
- **Supports Intel Object Module Format (OMF)**

PL/M 386 is a powerful, structured, high-level system implementation language for the development of system software for the protected virtual address mode 386. PL/M 386 has been enhanced to utilize 386 features—memory management and protection—for the implementation of multi-user, multi-tasking virtual memory operating systems.

PL/M 386 is upward compatible with PL/M 286 and PL/M 86. Existing systems software can be re-compiled with PL/M 386 to execute in protected virtual address mode on the 80386.

PL/M 386 is the high-level alternative to assembly language programming on the 80386. For the majority of 386 system programs, PL/M 386 provides the features needed to access and to control efficiently the underlying 386 hardware and consequently it is the cost-effective approach to develop reliable, maintainable system software.

The PL/M 386 compiler has been designed to efficiently support all phases of software development. Features such as a built-in syntax checker, multiple levels of optimization, virtual symbol table and four models of program size and memory usage for efficient code generation provide the total program development support needed.

FEATURES

Major features of the Intel PL/M 386 compiler and programming language include:

Structured Programming

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible by clearly defining the scope of user variables (local to a private procedure, for example).

The use of modules and procedures to break down a large problems leads to productive software development. The PL/M 386 implementation of block structure allows the use of REENTRANT procedures, which are especially useful in system design.

Language Compatibility

PL/M 386 object modules are compatible with object modules generated by all other 386 translators. This means that PL/M programs may be linked to programs written in any other 386 languages.

Object modules are compatible with In-Circuit Emulators; DEBUG compiler control provides the In-Circuit Emulators with full symbolic debugging capabilities.

PL/M 386 language is upward compatible with PL/M 286, PL/M 86 and PL/M 80 so that application programs may be easily ported to run on the protected mode 80386.

Supports Fourteen Data Types

PL/M makes use of fourteen data types for various applications. These data types range from one to eight bytes and facilitate various arithmetic, logic, and addressing functions:

—BIT(n):	1 to 32 bit unsigned number
—BYTE:	8 bits unsigned number
—HWORD:	16 bits unsigned number
—WORD:	32 bits unsigned number
—DWORD:	64 bits unsigned number
—OFFSET:	32 bits memory address
—CHARINT:	8 bits signed number
—SHORTINT:	16 bits signed number
—INTEGER:	32 bits signed number
—LONGINT:	64 bits signed number
—REAL:	32 bits floating-point number
—SELECTOR:	16 bits segment name
—POINTER:	48 bits selector, offset
—LONGREAL:	64 bits floating-point number

Another powerful facility allows the use of BASED variables which permit run-time mapping of variables to memory locations. This is especially useful for passing parameters, relative and absolute addressing, and dynamic memory allocation.

Data Type Compatibility

PL/M 286 programs may be recompiled and retargetted to the 386 by use of the WORD16 control. With this control, PL/M 386 provides transparent access to the seven data types provided by PL/M 286.

Two Data Structuring Facilities

In addition to the 14 data types and based variables, PL/M supports two powerful data structuring facilities. These help the user organize data into logical groups.

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of both: Arrays of structures or structures of arrays and structures within structures.

Numerics Support

PL/M programs that use 32-bit REAL data are executed using the 80287 Numeric Data Processor for high performance. All floating-point operations supported by PL/M are executed on the 80287 according to the IEEE floating-point standard. PL/M 386 programs can use built-in functions and predefined procedures—INIT$REAL$MATH$UNIT, SET$REAL $MODE, GET$REAL$ERROR, SAVE$REAL$ STATUS, RESTORE$REAL$STATUS—to control the operation of the 80287 within the scope of the language.

Built-In Port I/O

PL/M 386 directly supports input and output from the 386 ports for single BYTE, HWORD and WORD transfers. For BLOCK transfers, PL/M 386 programs can make calls to predefined procedures.

Interrupt Handling

PL/M 386 has the facility for generating and handling interrupts on the 386. A procedure may be defined as an interrupt handler through use of the INTERRUPT attribute. The compiler will then generate code to save and restore the processor status on each execution of the user-defined interrupt handler routine. The PL/M statement CAUSE$INTERRUPT allows the user to trigger a software interrupt from within the program.

Protection Model

PL/M 386 support the implementation of protected operating system software by providing built-in procedures and variables to access the protection mechanism of the 386. Predefined variables—TASK$REGISTER, LOCAL$TABLE, MACHINE$ STATUS, CONTROL$REGISTER, etc.—allow direct access and modification of the protection system. Untyped procedures and functions—SAVE$ GLOBAL$TABLE, RESTORE$GLOBAL$TABLE, SAVE$INTERRUPT$TABLE, RESTORE$INTERRUPT$TABLE, CLEAR$TASK$SWITCHED$FLAG, GET$ACCESS$RIGHTS, GET$SEGMENT$LIMIT, SEGMENT$READABLE, SEGMENT$WRITABLE, ADJUST$RPL—provide all the facilities needed to implement efficient operating system software.

Compiler Controls

The PL/M 386 compiler offers controls that facilitate such features as:

— Interface to other 386 languages
— Optimization
— Conditional compilation
— The inclusion of additional PL/M source files from disk
— Cross-reference of symbols
— Optional assembly language code in the listing file
— The setting of overflow conditions for run-time handling.
— WORD16/WORD32
— Interface to 286 languages

Addressing Control

The PL/M 386 compiler uses the SMALL and COMPACT controls to generate optimum addressing instructions for programs. Programs of any size can be easily modularized into "subsystems" to exploit the most efficient memory addressing schemes. This lowers total memory requirements and improves run-time execution of programs.

Code Optimization

The PL/M 386 compiler offers four levels of optimization for significantly reducing overall program size.

— Combination or "folding" of constant expressions; and short-circuit evaluation of Boolean expressions
— "Strength reductions": a shift left rather than multiply by 2; and elimination of common subexpressions within the same block
— Machine code optimizations; elimination of superfluous branches; removal of unreachable code
— Optimal local register allocation

Error Checking

The PL/M 386 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed and helpful set of programming and compilation error messages is provided by the compiler and user's guide.

Cost-Effective Alternative to Assembly Language

PL/M 386 programs are code efficient. PL/M 386 combines all of the benefits of a high-level language (ease of use, high productivity) with the ability to access the 386 architecture. Consequently, for the development of systems software, PL/M 386 is the cost-effective alternative to assembly language programming.

Support

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

ORDERING INFORMATION

Part Number	Description	Operating Environment
X286PLM386	PL/M 386 Compiler	XENIX* 286/310
D86PLM386	PL/M 386 Compiler	PC-DOS 3.0 or Greater
VVSPLM386	PL/M 386 Compiler	VAX/VMS 4.4 or Greater
MVVSPLM386	PL/M 386 Compiler	MicroVAX/VMS 4.4 or Greater

Documentation Package
PL/M 386 User's Guide for Xenix* 286 System

*XENIX is a trademark of Microsoft.

C 386
C COMPILER FOR THE 386

- **Implements full C Language**
- **Produces High Density Code Rivaling Assembler**
- **Supports Intel Object Module Format (OMF)**
- **Written in C**
- **Supports IEEE Floating Point Math with 80287 Coprocessor**
- **Supports Bit Fields**
- **Supports Full Standard I/O Library (STDIO)**
- **Complies with Draft Proposed ANSI C Standard**

Intel C 386 brings the full power of the C programming language to the 386 microprocessor system. Intel C386 supports the full C language as described in the Kernighan and Ritchie book, "The C Programming Language", (Prentice-Hall, 1978). Also included are the latest enhancements to the C language: structure assignments, functions taking structure arguments and returning structures, and the "void" and "enum" data types.

Intel C 386 Compiler Description

The C 386 compiler operates in several phases: preprocessor, parser and code generator. The preprocessor phase interprets directives in C source code, including conditional compilations (# define). The parser phase converts the C program into an intermediate free form and does all syntactic and semantic error checking. The code generator phase converts the parser's output into an efficient intermediate binary code, performs constant folding, and features an extremely efficient register allocator, ensuring high quality code. The code generator outputs relocatable Intel Object Module Format (OMF) code, without creating an intermediate assembly file. The C386 compiler eliminates common code, eliminates redundant loads and stores, and resolves span dependencies (shortens branches) within a program.

The C 386 runtime library consists of a number of functions which the C programmer can call. The runtime system includes the standard I/O library (STDIO), conversion routines, routines for manipulating strings, and (where appropriate) routines for interfacing with the operating system.

C 386 uses Intel's Binder and Builder and generates debug records for symbols and lines on request, permitting access to Intel's PSCOPE Monitor/ICE-386 emulator to aid in program testing.

FEATURES

Preprocessor Directives

#define—defines a macro

#include—includes code outside of the program source file

#if—conditionally includes or excludes code

Other preprocessor directives include #undef, #ifdef, #ifdef, #else, #endif, and #line.

Statements

The C language supports a variety of statements:

Conditionals: If, IF-ELSE

Loops: WHILE, DO-WHILE, FOR

Selection of cases: SWITCH, CASE, DEFAULT

Exit from a function: RETURN

Loop Control: CONTINUE, BREAK

Branching: GOTO

Expressions and Operators

The C language includes a rich set of expressions and operators.

Primary expression: invoke functions, select elements from arrays, and extract fields from structures or unions.

Arithmetic operators: add, subtract, multiply, divide, modulus

Relational operators: greater than, greater than or equal, less than, less than or equal, not equal

Unary operators: indirect through a pointer, compute an address, logical negation, ones complement, provide the size in bytes of an operand.

Logical operators: AND, OR

Bitwise operators: AND, exclusive OR, inclusive OR, bitwise complement

Data Types and Storage Classes

Data in C is described by its type and storage class. The type determines its representation and use, and the storage class determines its lifetime, scope, and storage allocation. The following data types are fully supported by C 386.

char: an 8 bit signed integer

int: a 32 bit signed integer

short: a 16 bit signed integer

long: a 32 bit signed integer

unsigned: a modifier for integer data types (char, int, short, and long) which doubles the positive range of values

float: a 32 bit floating point number which utilizes the 80287

double: a 64 bit floating point number

void: a special type that cannot be used as an operand in expressions; normally used for functions called only for effect (to prevent their use in contexts where a value is required).

enum: an enumerated data type

These fundamental data types may be used to create other data types including: arrays, functions, structures, pointers, and unions.

The storage classes available in C 386 include:

register: suggests that a variable be kept in a machine register, often enhancing code density and speed

extern: a variable defined outside of the function where it is declared; retaining its value throughout the entire program and accessible to other modules

auto: a local variable, created when a block of code is entered and discarded when the block is exited

static: a local variable that retains its value until the termination of the entire program

typedef: defines a new data type name from existing data types

BENEFITS

Faster Compilation

Intel C 386 compiles C programs substantially faster than standard C compilers because it produces Intel OMF code directly, eliminating the traditional intermediate process of generating an assembly file.

Portability of Code

Because Intel C 386 supports the STDIO and produces Intel OMF code, programs developed on a variety of machines can easily be transported to the 386.

Full Manipulation of the 386

Intel C 386 enables the programmer to utilize features of the C language to control bit fields, pointers, addresses and register allocation, taking full advantage of the fundamental concepts of the 386.

Support

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

ORDERING INFORMATION

Part Number	Description	Operating Environment
X286C386PP	C 386 Compiler	XENIX* 286/310 System
VVSC386	C 386 Compiler	VAX/VMS 4.3 and later
MVVSC386	C 386 Compiler	MicroVAX/VMS 4.4 or later
D86C386	C 386 Compiler	PC-DOS 3.0 or greater

Documentation Package

C 386 User's Guide for Xenix* 286 System

*XENIX is a trademark of Microsoft.

- **Library to Support Floating Point Arithmetic in Pascal-86, PL/M-86, FTN-86 and ASM-86**
- **Decimal Conversion Library Supports Binary-Decimal Conversions**
- **Supports Proposed IEEE Floating Point Standard for High Accuracy and Software Portability**
- **Common Elementary Function Library Provides Trigonometric, Logarithmic and Other Useful Functions**
- **Error-Handler Module Simplifies Floating Point Error Recovery**

The 8087 Support Library provides Pascal-86, FORTRAN-86, PL/M-86 and ASM-86 users with numeric data processing capability. With the Library, it is easy for programs to do floating point arithmetic. Programs can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs. Figure 1 below illustrates how the 8087 Support Library can be bound with PL/M-86 and ASM-86 user code to do this. The 8087 Support Library supports the proposed IEEE Floating Point Standard. Consequently, by using this Library, the user not only saves software development time, but is guaranteed that the numeric software meets industry standards and is portable–the software investment is maintained.

The 8087 Support Library consists of the common elementary function library (CEL87.LIB), the decimal conversion library (DC87.LIB), the emulator interface library E8087.LIB, the error handler module (EH87.LIB) and interface libraries (8087.LIB, NUL87.LIB).

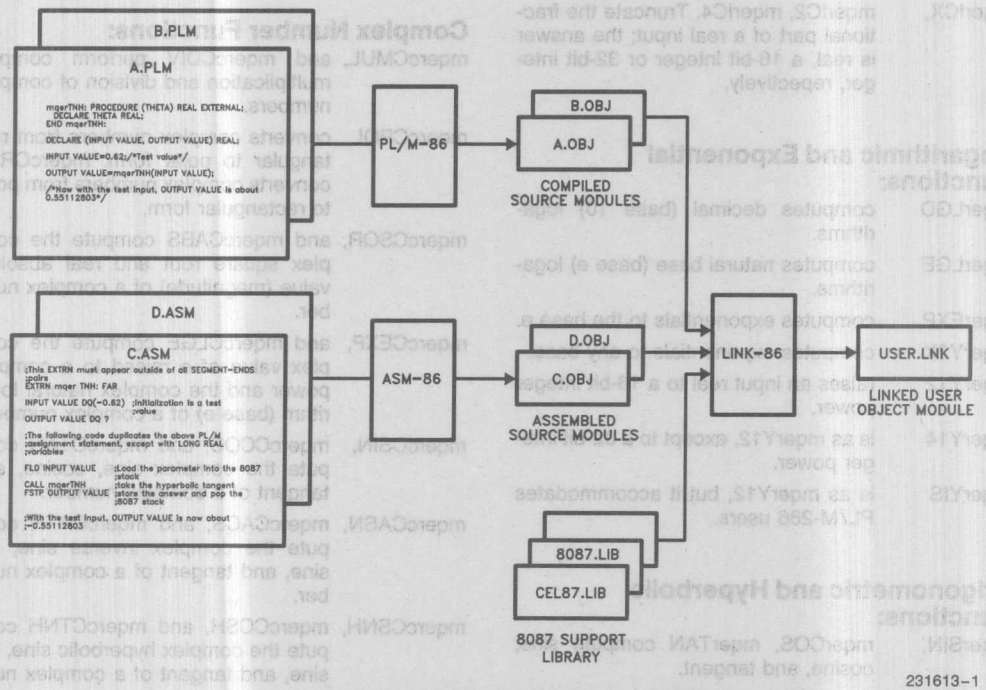


Figure 1. Use of 8087 Support Library with PL/M-86 and ASM-86

August 1985
Order Number: 231613-001

CEL87.LIB
THE COMMON ELEMENTARY FUNCTION LIBRARY

FUNCTIONS

CEL87.LIB contains commonly used floating point functions. It is used along with the 8087 numeric coprocessor. It provides a complete package of elementary functions, giving valid results for all appropriate inputs. Following is a summary of CEL87 functions, grouped by functionality.

Rounding and Truncation Functions:

mqerIEX, mqerIE2, and mqerIE4. Round a real number to the nearest integer; to the even integer if there is a tie. The answer returned is real, a 16-bit integer or a 32-bit integer respectively.

mqerIAX, mqerIA2, mqerIA4. Round a real number to the nearest integer, to the integer away from zero if there is a tie; the answer returned is real, a 16-bit integer or a 32-bit integer, respectively.

mqerICX, mqerIC2, mqerIC4. Truncate the fractional part of a real input; the answer is real, a 16-bit integer or 32-bit integer, repectively.

Logarithmic and Exponential Functions:

mqerLGD computes decimal (base 10) logarithms.

mqerLGE computes natural base (base e) logarithms.

mqerEXP computes exponentials to the base e.

mqerY2X computes exponentials to any base.

mqerY12 raises an input real to a 16-bit integer power.

mqerY14 is as mqerY12, except to a 32-bit integer power.

mqerYIS is as mqerY12, but it accommodates PL/M-286 users.

Trigonometric and Hyperbolic Functions:

mqerSIN, mqerCOS, mqerTAN compute sine, cosine, and tangent.

mqerASN, mqerACS, mqerATN compute the corresponding inverse functions.

mqerSNH, mqerCSH, mqerTNH compute the corresponding hyperbolic functions.

mqerAT2 is a special version of the arc tangent function that accepts rectangular coordinate inputs.

Other Functions (of real variables):

mqerDIM is FORTRAN's positive difference function.

mqerMAX returns the maximum of two real inputs.

mqerMIN returns the minimum of two real inputs.

mqerSGH combines the sign of one input with the magnitude of the other input.

mqerMOD computes a modulus, retaining the sign of the dividend.

mqerRMD computes a modulus, giving the value closest to zero.

Complex Number Functions:

mqercCMUL, and mqercCDIV perform complex multiplication and division of complex numbers.

mqercCPOL converts complex numbers from rectangular to polar form. mqercCREC converts complex numbers from polar to rectangular form.

mqercCSQR, and mqercCABS compute the complex square root and real absolute value (magnitude) of a complex number.

mqercCEXP, and mqercCLGE compute the complex value of e raised to a complex power and the complex natural logarithm (base e) of a complex number.

mqercCSIN, mqercCCOS, and mqercCTAN compute the complex sine, cosine, and tangent of a complex number.

mqercCASN, mqercCACS, and mqercCATN compute the complex inverse sine, cosine, and tangent of a complex number.

mqercCSNH, mqercCCSH, and mqercCTNH compute the complex hyperbolic sine, cosine, and tangent of a complex number.

mqercCACH, mqercCASH, and mqercCATH compute the comples inverse hyperbolic sine, cosine, and tangent of a complex number.

mqercCC2C, mqercCR2C, mqercCC2R, mqercCCI2, mqercCCI4, and mqercCCIS return complex values of complex (or real) values raised to complex (real, short integer, or long integer) values.

DC87.LIB
THE DECIMAL CONVERSION LIBRARY

DC87.LIB is a library of procedures which convert binary representations of floating point numbers and ASCII-encoded string of digits.

The binary-to-decimal procedure mqcBIN__DECLOW accepts a binary number in any of the formats used for the representation of floating point numbers in the 8087. Because there are so many output formats for floating point numbers, mqcBIN__DECLOW does not attempt to provide a finished, formatted text string. Instead, it provides the "building blocks" for you to use to construct the output string which meets your exact format specification.

The decimal-to-binary procedure mqcDEC__BIN accepts a text string which consists of a decimal number with optional sign, decimal point, and/or power-of-ten exponent. It translates the string into the caller's choice of binary formats.

Decimal-to-binary procedure mqcDECLOW__BIN is provided for callers who have already broken the decimal number into its constituent parts.

The procedures mqcLONG__TEMP, mqcSHORT__TEMP, mqcTEMP__LONG, and mqcTEMP__SHORT convert floating point numbers between the longest binary format, TEMP__REAL, and the shorter formats.

EH87.LIB
THE ERROR HANDLER LIBRARY

EH87.LIB is a library of five utility procedures for writing trap handlers. Trap handlers are called when an unmasked 8087 error occurs.

The 8087 error reporting mechanism can be used not only to report error conditions, but also to let software implement IEEE standard options not directly supported by the chip. The three such extensions to the 8087 are: normalizing mode, non-trapping not-a-number (NaN), and non-ordered comparison. The utility procedures support these extra features.

DECODE is called near the beginning of the trap handler. It preserves the complete state of the 8087, and also identifies what function called the trap handler, and returns available arguments and/or results. DECODE eliminates much of the effort needed to determine what error caused the trap handler to be called.

NORMAL provides the "normalizing mode" capability for handling the "D" exception. By calling NORMAL in your trap handler, you eliminate the need to write code in your application program which tests for non-normal inputs.

SIEVE provides two capabilities for handling the "I" exception. It implements non-trapping NaN's and non-ordered comparisons. These two IEEE standard features are useful for diagnostic work.

ENCODE is called near the end of the trap handler. It restores the state of the 8087 saved by DECODE, and performs a choice of concluding actions, by either retrying the offending function or returning a specified result.

FILTER calls each of the above four procedures. If your error handler does nothing more than detect fatal errors and implement the features supported by SIEVE and NORMAL, then your interface to EH87.LIB can be accomplished with a single call to FILTER.

8087.LIB, NUL87.LIB, E8087.LIB INTERFACE LIBRARIES

E8087.LIB, 8087.LIB and NUL87.LIB libraries configure a user's application program for his run-time environment; running with the 8087 component or without floating point arithmetic, respectively.

FULL 8087 EMULATOR

The Full 8087 Emulator is a 16-kilobyte object module that is linked to the application program for floating-point operations. Its functionality is identical to the 8087 chip, and is ideal for prototyping and debugging floating-point applications. The Emulator is an alternative to the use of the 8087 chip, although the latter executes floating-point applications up to 100 times faster than an 8086 with the 8087 Emulator. Furthermore, since the 8087 is a "coprocessor," use of the chip will allow many operations to be performed in parallel with the 8086.

SPECIFICATIONS

Operating Environment

Intel Microcomputer Development Systems (Series III, Series IV)

Documentation Package

8087 Support Library Reference Manual

ORDERING INFORMATION

Part Number	Description
iMDS 319	8087 Support Library

Requires Software License

SUPPORT

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

80287 SUPPORT LIBRARY

- **Library to support floating point arithmetic in Pascal-286, PL/M-286 and ASM-286**
- **Decimal conversion library supports binary-decimal conversions**
- **Supports proposed IEEE Floating Point Standard for high accuracy and software portability**
- **Common elementary function library provides trigonometric, logarithmic and other useful functions**
- **Error-handler module simplifies floating point error recovery**

The 80287 Support Library provides Pascal-286, PL/M-286 and ASM-286 users with numeric data processing capability. With the Library, it is easy for programs to do floating point arithmetic. Programs can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs. Figure 1 below illustrates how the 80287 Support Library can be bound with PL/M-286 and ASM-286 user code to do this. The 80287 Support Library supports the proposed IEEE Floating Point Standard. Consequently, by using this Library, the user not only saves software development time, but is guaranteed that the numeric software meets industry standards and is portable–the software investment is maintained.

The 80287 Support Library consists of the common elementary function library (CEL287.LIB), the decimal conversion library (DC287.LIB), the error handler module (EH287.LIB) and interface libraries (80287.LIB, NUL287.LIB).

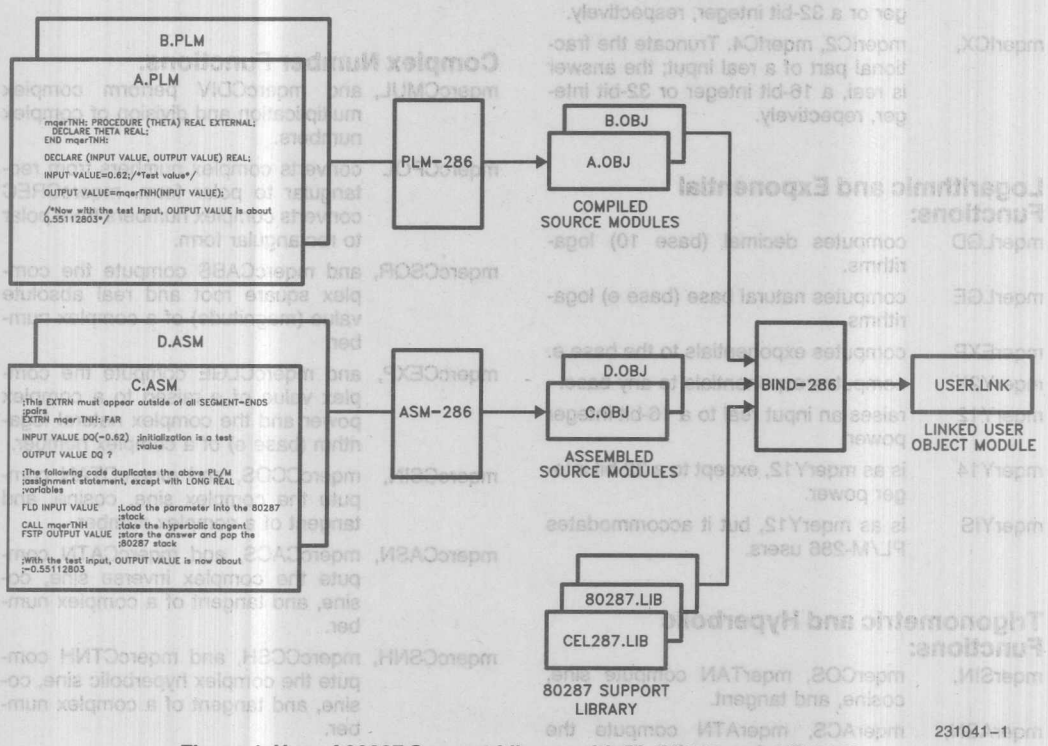


Figure 1. Use of 80287 Support Library with PL/M-286 and ASM-286

October 1986
Order Number: 231041-002

CEL287.LIB
THE COMMON ELEMENTARY FUNCTION LIBRARY

FUNCTIONS

CEL287.LIB contains commonly used floating point functions. It is used along with the 80287 numeric coprocessor. It provides a complete package of elementary functions, giving valid results for all appropriate inputs. Following is a summary of CEL287 functions, grouped by functionality.

Rounding and Truncation Functions:

mqerIEX, mqerIE2, and mqerIE4. Round a real number to the nearest integer; to the even integer if there is a tie. The answer returned is real, a 16-bit integer or a 32-bit integer respectively.

mqerIAX, mqerIA2, mqerIA4. Round a real number to the nearest integer, to the integer away from zero if there is a tie; the answer returned is real, a 16-bit integer or a 32-bit integer, respectively.

mqerICX, mqerIC2, mqerIC4. Truncate the fractional part of a real input; the answer is real, a 16-bit integer or 32-bit integer, repectively.

Logarithmic and Exponential Functions:

mqerLGD computes decimal (base 10) logarithms.

mqerLGE computes natural base (base e) logarithms.

mqerEXP computes exponentials to the base e.

mqerY2X computes exponentials to any base.

mqerY12 raises an input real to a 16-bit integer power.

mqerY14 is as mqerY12, except to a 32-bit integer power.

mqerYIS is as mqerY12, but it accommodates PL/M-286 users.

Trigonometric and Hyperbolic Functions:

mqerSIN, mqerCOS, mqerTAN compute sine, cosine, and tangent.

mqerASN, mqerACS, mqerATN compute the corresponding inverse functions.

mqerSNH, mqerCSH, mqerTNH compute the corresponding hyperbolic functions.

mqerAT2 is a special version of the arc tangent function that accepts rectangular coordinate inputs.

Other Functions (of real variables):

mqerDIM is FORTRAN's positive difference function.

mqerMAX returns the maximum of two real inputs.

mqerMIN returns the minimum of two real inputs.

mqerSGH combines the sign of one input with the magnitude of the other input.

mqerMOD computes a modulus, retaining the sign of the dividend.

mqerRMD computes a modulus, giving the value closest to zero.

Complex Number Functions:

mqercCMUL, and mqercCDIV perform complex multiplication and division of complex numbers.

mqercCPOL converts complex numbers from rectangular to polar form. mqercCREC converts complex numbers from polar to rectangular form.

mqercCSQR, and mqercCABS compute the complex square root and real absolute value (magnitude) of a complex number.

mqercCEXP, and mqercCLGE compute the complex value of e raised to a complex power and the complex natural logarithm (base e) of a complex number.

mqercCSIN, mqercCCOS, and mqercCTAN compute the complex sine, cosine, and tangent of a complex number.

mqercCASN, mqercCACS, and mqercCATN compute the complex inverse sine, cosine, and tangent of a complex number.

mqercCSNH, mqercCCSH, and mqercCTNH compute the complex hyperbolic sine, cosine, and tangent of a complex number.

Complex Number Functions: (Continued)

mqercCACH, mqercCASH, and mqercCATH compute the comples inverse hyperbolic sine, cosine, and tangent of a complex number.

mqercCC2C, mqercCR2C, mqercCC2R, mqercCCI2, mqercCCI4, and mqercCCIS return complex values of complex (or real) values raised to complex (real, short integer, or long integer) values.

DC287.LIB
THE DECIMAL CONVERSION LIBRARY

DC287.LIB is a library of procedures which convert binary representations of floating point numbers and ASCII-encoded string of digits.

The binary-to-decimal procedure mqcBIN__DECLOW accepts a binary number in any of the formats used for the representation of floating point numbers in the 80287. Because there are so many output formats for floating point numbers, mqcBIN__DECLOW does not attempt to provide a finished, formatted text string. Instead, it provides the "building blocks" for you to use to construct the output string which meets your exact format specification.

The decimal-to-binary procedure mqcDEC__BIN accepts a text string which consists of a decimal number with optional sign, decimal point, and/or power-of-ten exponent. It translates the string into the caller's choice of binary formats.

Decimal-to-binary procedure mqcDECLOW__BIN is provided for callers who have already broken the decimal number into its constituent parts.

The procedures mqcLONG__TEMP, mqcSHORT__TEMP, mqcTEMP__LONG, and mqcTEMP__SHORT convert floating point numbers between the longest binary format, TEMP__REAL, and the shorter formats.

EH287.LIB
THE ERROR HANDLER LIBRARY

EH287.LIB is a library of five utility procedures for writing trap handlers. Trap handlers are called when an unmasked 80287 error occurs.

The 80287 error reporting mechanism can be used not only to report error conditions, but also to let software implement IEEE standard options not directly supported by the chip. The three such extensions to the 80287 are: normalizing mode, non-trapping not-a-number (NaN), and non-ordered comparison. The utility procedures support these extra features.

DECODE is called near the beginning of the trap handler. It preserves the complete state of the 80287, and also identifies what function called the trap handler, and returns available arguments and/or results. DECODE eliminates much of the effort needed to determine what error caused the trap handler to be called.

NORMAL provides the "normalizing mode" capability for handling the "D" exception. By calling NORMAL in your trap handler, you eliminate the need to write code in your application program which tests for non-normal inputs.

SIEVE provides two capabilities for handling the "I" exception. It implements non-trapping NaN's and non-ordered comparisons. These two IEEE standard features are useful for diagnostic work.

ENCODE is called near the end of the trap handler. It restores the state of the 80287 saved by DECODE, and performs a choice of concluding actions, by either retrying the offending function or returning a specified result.

FILTER calls each of the above four procedures. If your error handler does nothing more than detect fatal errors and implement the features supported by SIEVE and NORMAL, then your interface to EH287.LIB can be accomplished with a single call to FILTER.

80287.LIB, NUL287.LIB INTERFACE LIBRARIES

80287.LIB and NUL287.LIB libraries configure a user's application program for his run-time environment; running with the 80287 component or without floating point arithmetic, respectively.

SPECIFICATIONS

Operating Environment

Intel Microcomputer Development Systems (Series III, Series IV)

Documentation Package

80287 Support Library Reference Manual

Related Software

A 80287 software emulator is available as part of the 8086 software toolbox (iMDX364)

ORDERING INFORMATION

Part Number	Description
iMDX329	80287 Support Library

Requires Software License

SUPPORT

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

VAX*/VMS* RESIDENT 8086/88/186 SOFTWARE DEVELOPMENT PACKAGES

- **Executes on DEC VAX*/MicroVAX Minicomputer under VMS* Operating System to translate PL/M-86, Pascal-86 and ASM-86 Programs for 8086, 88 and 186 Microprocessors.**
- **Packages include C-86; FORTRAN-86; Pascal-86; PL/M-86; ASM-86; Link and Relocation Utilities; OH-86 Absolute Object Module to Hexadecimal Format Converter; and Library Manager Program.**
- **Output linkable with Code Generated on Intellec® Development Systems.**

The VAX/VMS Resident Software Development Packages contain software development tools for the 8086, 88, and 186 microprocessors. The package lets the user develop, compile, maintain libraries, and link and locate programs on a VAX running the VMS operating system. The translator output is object module compatible with programs translated by the corresponding version of the translator on an Intellec Development System.

Four packages are available:

1. An ASM-86 Assembler Package which includes the Assembler, the Link Utility, the Locate Utility, the absolute object to hexadecimal format conversion utility and the Library Manager Program.
2. A PL/M-86 Compiler Package which contains the PL/M-86 Compiler and Runtime Support Libraries.
3. A Pascal-86 Compiler Package which contains the Pascal-86 Compiler and Runtime Support Libraries.
4. A C-86 Compiler Package which contains the C-86 Compiler and Run-Time Libraries.
5. A FORTRAN-86 Compiler Package which contains the FORTRAN-86 Compiler and Run-Time Libraries.

The VAX/VMS resident development packages and the Intellec Development System development packages are built from the same technology base. Therefore, the VAX/VMS resident development packages and the Intellec Development System development packages are very similar.

Version numbers can be used to identify features correspondence. The VAX/VMS resident development packages will have the same features as the Intellec Development System product with the same version number.

The object modules produced by the translators contain symbol and type information for programming debugging using ICE™ translators and/or the PSCOPE debugger. For final production version, the compiler can remove this extra information and code.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

September 1987
Order Number: 210643-004

VAX*-PL/M-86/88/186 SOFTWARE PACKAGE

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Supports 16-Bit Signed Integer and 32-Bit Floating Point Arithmetic in Accordance with IEEE Proposed Standard**
- **Easy-To-Learn Block-Structured Language Encourages Program Modularity**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086 Object Modules, Generated on Either a VAX*, a PC XT/AT running PC-DOS Version 3.0 or Intellec® Development Systems**
- **Code Optimization Assures Efficient Code Generation and Minimum Application Memory Utilization**
- **Built-In Syntax Checker Doubles Performance for Compiling Programs Containing Errors**
- **Source Input/Object Output Compatible with PL/M-86 Hosted on an Intellec® Development System**
- **ICE™, PSCOPE Symbolic Debugging Fully Supported**

Like its counterpart for MCS®-80/85 program development, and Intellec® hosted 8086 program development, VAX-PL/M-86 is an advanced, structured high-level programming language. The VAX-PL/M-86 compiler was created specifically for performing software development for the Intel 8086, 88 and 186 Microprocessors.

PL/M is a powerful, structured, high-level system implementation language in which program statements can naturally express the program algorithm. This frees the programmer to concentrate on the logic of the program without concern for burdensome details of machine or assembly language programming (such as register allocation, meanings of assembler mnemonics, etc.).

The VAX-PL/M-86 compiler efficiently converts free-form PL/M language statements into equivalent 8086/88/186 machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

The use of PL/M high-level language for system programming, instead of assembly language, results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-on maintenance costs for the user.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-PASCAL-86/88 SOFTWARE PACKAGE

- **Executes VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086 Object Modules, Generated on Either a VAX*, a PC XT/AT running PC-DOS Version 3.0 or Intellec® Development Systems**
- **ICE™, PSCOPE Symbolic Debugging Fully Supported**
- **Implements REALMATH for Consistent and Reliable Results**
- **Supports 8086/20, 88/20 Numeric Data Processors**
- **Strict Implementation of ISO Standard Pascal**
- **Useful Extensions Essential for Microcomputer Applications**
- **Separate Compilation with Type-Checking Enforced between Pascal Modules**
- **Compiler Option to Support Full Run-Time Range-Checking**
- **Source Input/Object Output Compatible with Pascal-86 Hosted on a Intellec® Development System**

VAX-PASCAL-86 conforms to and implements the ISO Pascal standard. The language is enhanced to support microcomputer applications with special features, such as separate compilation, interrupt handling and direct port I/O. Other extensions include additional data types not required by the standard and miscellaneous enhancements such as an allowed underscore in names, an OTHERWISE clause in CASE construction and so forth. To assist the development of portable software, the compiler can be directed to flag all non-standard features.

The VAX-PASCAL-86 compiler runs on the Digital Equipment Corporation VAX under the VMS Operating System. A well-defined I/O interface is provided for run-time support. This allows a user-written operating system to support application programs on the target system as an alternate to the development system environment. Program modules compiled under PASCAL-86 are compatible and linkable with modules written in PL/M-86, and ASM-86. With a complete family of compatible programming languages for the 8086, 88, and 186 one can implement each module in the language most appropriate to the task at hand.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX* 8086/88/186 MACRO ASSEMBLER

- **Executes on VAX*/MicroVAX Minicomputers under The VMS* Operating System**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086/88/186 Object Modules, Generated on Either a VAX*, a PC XT/AT running PC-DOS Version 3.0 or Intellec® Development Systems**
- **Powerful and Flexible Text Macro Facility with Three Macro Listing Options to Aid Debugging**
- **Highly Mnemonic and Compact Language, Most Mnemonics Represent Several Distinct Machine Instructions**
- **"Strongly Typed" Assembler Helps Detect Errors at Assembly Time**
- **High-Level Data Structuring Facilities Such as "STRUCTURES" and "RECORDS"**
- **Over 120 Detailed and Fully Documented Error Messages**
- **Produces Relocatable and Linkable Object Code**
- **Source Input/Object Output Compatible with ASM-86 hosted on an Intellec® Development System**

VAX-ASM-86 is the "high-level" macro assembler for the 8086/88/186 assembly language. VAX-ASM-86 translates symbolic 8086/88/186 assembly language mnemonics into 8086/88/186 relocatable object code.

VAX-ASM-86 should be used where maximum code efficiency and hardware control is needed. The 8086/88/186 assembly language includes approximately 100 instruction mnemonics. From these few mnemonics the assembler can generate over 3,800 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 100 mnemonics to generate all possible 8086/88/186 machine instructions. VAX-ASM-86 will generate the shortest machine instruction possible given no forward referencing or given explicit information as to the characteristics of forward referenced symbols.

VAX-ASM-86 offers many features normally found only in high-level languages. The 8086/88/186 assembly language is strongly typed. The assembler performs extensive checks on the usage of variable and labels. The assembler uses the attributes which are derived explicity when a variable or label is first defined, then makes sure that each use of the symbol in later instructions conforms to the usage defined for that symbol. This means that many programming errors will be deteced when the program is assembled, long before it is being debugged on hardware.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-LIB-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **VAX-LIB-86 is a Library Manager Program which Allows You to:**
  Create Specifically Formatted Files to Contain Libraries of Object Modules
  Maintain These Libraries by Adding or Deleting Modules
  Print a Listing of the Modules and Public Symbols in a Library File
- **Libraries Can be Used as Input to VAX-LINK-86 Which Will Automatically Link Modules from the Library that Satisfy External References in the Modules Being Linked**
- **Abbreviated Control Syntax**

Libraries aid in the job of building programs. The library manager program VAX-LIB-86 creates and maintains files containing object modules. The operation of VAX-LIB-86 is controlled by commands to indicate which operation VAX-LIB-86 is to perform. The commands are:

CREATE:	creates an empty library file
ADD:	adds object modules to a library file
DELETE:	deletes modules from a library file
LIST:	lists the module directory of library files
EXIT:	terminates the LIB-86 program and returns control to VMS

When using object libraries, the linker will call only those object modules that are required to satisfy external references, thus saving memory space.

VAX-OH-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Converts an 8086/88/186 Absolute Object Module to Symbolic Hexademical Format**
- **Facilitates Preparing a file for Loading by Symbolic Hexadecimal Loader (e.g. iSBC® Monitor SDK-86 Loader), or Universal PROM Mapper**
- **Converts an Absolute Module to a More Readable Format that can be Displayed on a CRT or Printed for Debugging**

The VAX-OH-86 utility converts an 86/88 absolute object module to the hexadecimal format. This conversion may be necessary for later loading by a hexadecimal loader such as the iSBC 86/12 monitor or the Universal PROM Mapper. The conversion may also be made to put the module in a more readable format that can be displayed or printed.

The module to be converted must be in absolute form; the output from VAX-LOC-86 is in absolute format.

*VAX, VMS are trademarks of Digital Equipment Corporation.

VAX*-LINK-86

- **Executes on VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Automatic Combination of Separately Compiled or Assembled 86/88/186 Programs into a Relocatable Module, Generated on Either a VAX, a PC XT/AT running PC-DOS Version 3.0 or an Intellec® Development System**
- **Automatic Selection of Required Modules from Specified Libraries to Satisfy Symbolic References**
- **Extensive Debug Symbol Manipulation, allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**
- **Automatic Generation of a Summary Map Giving Results of the LINK-86 Process**
- **Abbreviated Control Syntax**
- **Relocatable modules may be Merged into a Single Module Suitable for Inclusion in a Library**
- **Supports "Incremental" Linking**
- **Supports Type Checking of Public and External Symbols**

VAX-LINK-86 combines object modules specified in the VAX-LINK-86 input list into a single output module. VAX-LINK-86 combines segments from the input modules according to the order in which the modules are listed.

VAX-LINK-86 will accept libraries and object modules built from VAX-PL/M-86, VAX-PASCAL-86, VAX-ASM-86, or any other Intel translator generating 8086 Relocatable Object Modules, such as the Series III resident translators.

Support for incremental linking is provided since an output module produced by VAX-LINK-86 can be an input to another link. At each stage in the incremental linking process, unneeded public symbols may be purged.

VAX-LINK-86 supports type checking of PUBLIC and EXTERNAL symbols reporting a warning if their types are not consistent.

VAX-LINK-86 will link any valid set of input modules without any controls. However, controls are available to control the output of diagnostic information in the VAX-LINK-86 process and to control the content of the output module.

VAX-LINK-86 allows the user to create a large program as the combination of several smaller, separately compiled modules. After development and debugging of these component modules the user can link them together, locate them using VAX-LOC-86 and enter final testing with much of the work accomplished.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

VAX*-LOC-86

- **Executes on the VAX*/MicroVAX Minicomputers under the VMS* Operating System**
- **Automatic Generation of a Summary Map Giving Starting Address, Segment Addresses and Length, and Debug Symbols and their Addresses**
- **Extensive Capability to Manipulate the Order and Placement of Segments in 8086/8088 Memory**
- **Abbreviated Control Syntax**
- **Automatic and Independent Relocation of Independent Relocation of Segments. Segments May be Relocated to Best Match Users Memory Configuration**
- **Extensive Debug Symbol Manipulation, Allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**

Relocatability allows the programmer to code programs or sections of programs without having to know the final arrangement of the object code in memory.

VAX-LOC-86 converts relative addresses in an input module in iAPX-86/88/186 object module format to absolute addresses. VAX-LOC-86 orders the segments in the input module and assigns absolute addresses to the segments. The sequence in which the segments in the input module are assigned absolute addresses is determined by their order in the input module and the controls supplied with the command.

VAX-LOC-86 will relocate any valid input module without any controls. However, controls are available to control the output of diagnostic information in the VAX-LOC-86 process, to control the content of the output module, or both.

The program you are developing will almost certainly use some mix of random access memory (RAM), read-only memory (ROM), and/or programmable read-only memory (PROM). Therefore, the location of your program affects both cost and performance in your application. The relocation feature allows you to develop your program and then simply relocate the object code to suit your application.

*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.

SPECIFICATIONS

Operating Environment

Required Hardware

VAX* 11/780, 11/782, 11/750, or 11/730 9 Track Magnetic Tape Drive, 1600 BPI

MicroVAX II with TK-50 tape drive.

Required Software

VMS Operating System V3.0 or Later. All of the development packages are delivered as unlinked VAX object code which can be linked to VMS as designed for the system where the development package is to be used. VMS command files to perform the link are provided.

MicroVMS (V4.4 or later).

Documentation Package

iAPX-86, 88 Development Software Installation Manual and User's Guide for VAX/VMS, Order number 121950-001

Shipping Media

9 Track Magnetic Tape 1600 bpi (VAX)

TK-50 Cartridge Tape (MicroVAX)

ORDERING INFORMATION

Part Number	Description
VVSASM86	VAX-ASM-86, VAX-LINK-86, VAX-LOC-86, VAX-LIB-86, VAX-OH-86, Package
VVSPLM86	VAX-PLM-86 Package
iMDX-344VX	VAX-PASCAL-86 Package
VVSC86	VAX-C-86 Package
MVVSASM86	MICROVAX ASM86 Package
MVVSPLM86	MICROVAX PLM86 Package
MVVSC86	MICROVAX C86 Package
MVVSFORT86	MICROVAX FORTRAN 86 Package

REQUIRES SOFTWARE LICENSE

VAX*/VMS* RESIDENT SOFTWARE DEVELOPMENT PACKAGES FOR 80286

- **Hosted on DEC VAX*/MicroVAX Minicomputers Under the VMS* Operating System**
- **Allows Development of System and Application Software for the Protected Virtual Address Mode of the 80286**
- **Packages include ASM-286, PL/M-286, BUILD-286, BIND-286, LIB-286, MAP-286 and C-286**
- **Compatible with Corresponding Intel Development System Resident Products**

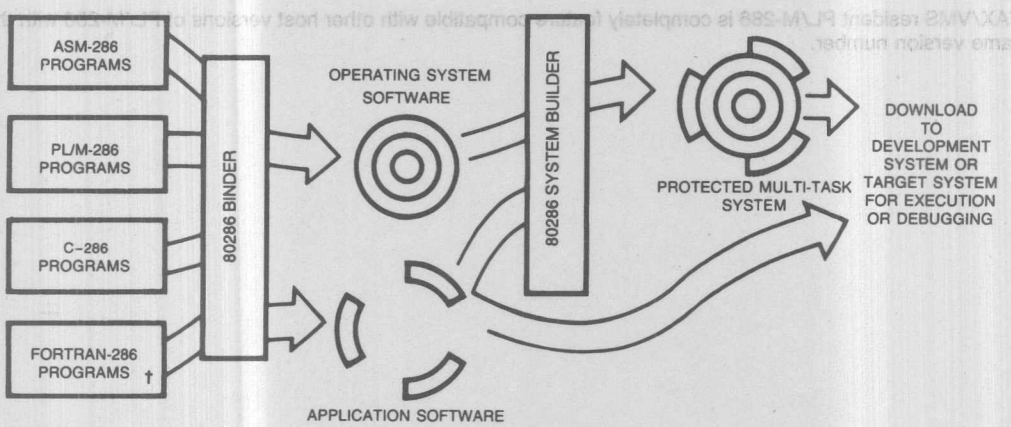
These packages provide the capability of developing software on a VAX*/VMS* host for the 80286 in protected virtual address mode. With these packages a user can assemble and compile 286 programs, configure system and application software and create and manage 286 object libraries. Figure 1 illustrates the process of 286 software development on VAX*/VMS* hosts.

Two packages are available:

1. A PL/M-286 package which contains the PL/M-286 compiler and run time support libraries.
2. An ASM-286 package which contains the 80286 Assembler (ASM-286) and programming utilities. These utilities include the 80286 System Builder (BLD-286), the System Binder (BND-286), a Library Utility (LIB-286) and an Object Map Utility (MAP-286).

These packages are compatible with corresponding products which are hosted on PC-DOS and Intel development systems. Correspondence can be established via version numbers. For example, BND-286 V2.0 offers the same set of features on VAX/VMS and Intel development systems.

Owing to this compatibility, 80286 software developed on VAX/VMS can be linked to 80286 software from development systems. Moreover, 80286 programs developed on the VAX can then be downloaded to development systems and debugged using 286 debuggers like the I²ICE™-286 system.



231038-1

Figure 1. 286 Software Development on VAX*/VMS*

*VAX, VMS are trademarks of Digital Equipment Corporation

†Currently Available on Intel Development Systems Only

September 1987
Order Number: 231038-003

VAX*/VMS* RESIDENT PL/M-286

- **Hosted on DEC VAX*/MicroVAX Minicomputers Under the VMS* Operating System**
- **Systems Programming Language for the Protected Virtual Address Mode 80286**
- **Enhanced to Support Design of Protected, Multi-User, Multi-Tasking, Virtual Memory Operating System Software**
- **Provides Multiple Levels of Optimization to Produce Efficient Code**
- **Produces Relocatable Object Code Linkable to Object Modules Generated by Other Intel 286 Language Translators**
- **Upward Compatible with PL/M-86 and PL/M-80 to Allow Software Portability**
- **Compatible with Development System Resident PL/M-286**

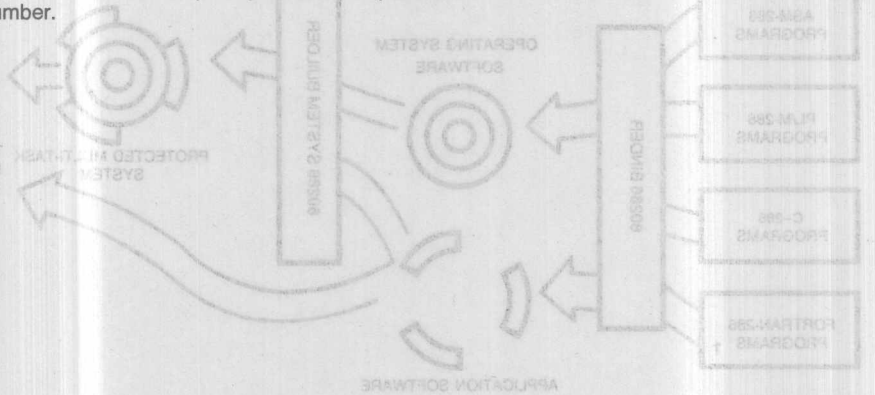
PL/M-286 is a powerful, structured, high-level system implementation language for the development of system software for the protected virtual address mode 80286. PL/M-286 has been enhanced to utilize 80286 features-memory management and protection–for the implementation of multi-user, multi-tasking virtual memory operating systems.

PL/M-286 is upward compatible with PL/M-86 and PL/M-80. Existing systems software can be re-compiled with PL/M-286 to execute in protected virtual address mode on the 80286.

PL/M-286 is the high-level alternative to assembly language programming on the 80286. For the majority of 80286 system programs, PL/M-286 provides the features needed to access and to control efficiently the underlying 80286 hardware, and consequently it is the cost-effective approach to develop reliable, maintainable system software.

The PL/M-286 compiler has been designed to efficiently support all phases of software development. Features such as built-in syntax checker, multiple levels of optimization, virtual symbol table and four models of program size and memory usage for efficient code generation provide the total program development support needed. The compiler also provides complete symbolic debug capability to the various 286 debuggers and emulators.

VAX/VMS resident PL/M-286 is completely feature compatible with other host versions of PL/M-286 with the same version number.



VAX*/VMS* RESIDENT 80286 MACRO ASSEMBLER

- **Supports Full Instruction Set of the 80286 including Memory Protection and Numerics (with 80287)**
- **Structures and RECORDS Provide Powerful Data Representation**
- **Type Checking at Assembly Time Helps Reduce Errors at Run-Time**
- **Powerful and Flexible Text Macro Facility**
- **Upward Compatible with ASM-86/88/186**
- **Compatible with Development System Resident 80286 Macro Assembler**

ASM-286 is the "high-level" macro assembler for the 80286 assembly language. ASM-286 translates symbolic assembly language mnemonics into relocatable object code. The assembler mnemonics are a superset of ASM-86/88 mnemonics; new ones have also been added to support the new 80286 instructions. The segmentation directives have been greatly simplified.

The 80286 assembly language includes approximately 150 instruction mnemonics. From these few mnemonics the assembler can generate over 4,000 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 150 mnemonics to generate all possible machine instructions. ASM-286 generates the shortest machine instruction possible (given explicit information as to the characteristics of any forward referenced symbols).

The powerful macro facility in ASM-286 saves development and maintenance time by coding common program sequences only once. A macro substitution is made each time the sequence is to be used. This facility also allows for conditional assembly of certain program sequences.

ASM-286 offers many features normally found only in high-level languages. The assembly language is strongly typed, which means it performs extensive checks on the usage of variables and labels. This means that many programming errors will be detected when the program is assembled, long before it is being debugged.

ASM-286 object modules conform to a thorough, well-defined format used by 286 high-level languages and utilities. This makes it easy to call (and be called from) HLL object modules.

ASM-286 also provides support for the 80287 numerics co-processor. The complete instruction set of the 80287 is available through high-level mnemonics.

VAX/VMS resident ASM-286 is completely feature compatible with other host versions of ASM-286 with the same version number.

VAX*/VMS* RESIDENT 80286 SYSTEM BUILDER

- **A Tool for Configuring Multi-Tasking Protected, Virtual Memory Systems Software for the 80286**
- **Links Separately Compiled Modules Resolves EXTERNAL/PUBLIC Definitions**
- **Creates a Memory Image of a 286 System for Cold Start Execution**
- **Target System May Be Bootloadable, Programmed into ROM or Loaded from Mass Storage**
- **Generates Print File with Command Listing and System Map**
- **Compatible with Development System Resident 80286 System Builder**

BLD-286 is the 80286 System Builder. It allows systems programmers to configure multi-tasking and memory protected 80286 software. The configuration is specified by the user in a "Build file" using a symbolic meta-language. BLD-286 thus provides the programmer a high-level symbolic interface to the multi-tasking and memory protection features of the 80286 architecture.

BLD-286 accepts as inputs object modules from the 80286 translators, the 80286 Binder and itself (for incremental building). Using the programmer's specifications in the Build File, it produces a bootloadable or loadable module as well as a print file with a map of the configured module.

Using the builders command language, system programmers may perform the following functions:

— Assign physical addresses to segments; also set segment access rights and limits.
— Create Call, Trap, and Interrupt "Gates" (entry-points) for inter-level program transfers.
— Make gates available to tasks; this is an easier way to define program interfaces than using interface libraries.
— Create Global (GDT), Interrupt (IDT), and any Local (LDT) Descriptor Tables.
— Create Task State Segments and Task Gates for multi-tasking applications.
— Resolve inter-module and inter-level references, and perform type-checking.
— Automatically select required modules from libraries.
— Configure the memory image into partitions in the address space.
— Selectively generate an object file and various sections of the print file.

VAX/VMS BLD-286 is completely feature compatible with other host versions of BLD-286 with the same version number.

VAX*/VMS* RESIDENT 80286 BINDER

- **Links Separately Compiled Program Modules into an Executable Task**
- **Makes the 80286 Protection Mechanism Invisible to Application Programmers**
- **Assigns Virtual Addresses to Tasks**
- **Performs Incremental Linking with Output of Binder and Builder**
- **Resolves PUBLIC/EXTERNAL Code and Data References, and Performs Intermodule Type-Checking**
- **Provides Print File Showing Segment Map, Errors and Warnings**
- **Generates Linkable or Loadable Module for Debugging**
- **Compatible with Development System Resident 80286 Binder**

BND-286 is a utility that combines 80286 object modules into executable tasks. In creating a task, the Binder resolves Public and External symbol references, combines segments, and performs address fix-ups on symbolic code and data.

The Binder takes object modules, produced by the 286 translators, and generates a loadable module (for execution or debugging), or a linkable module (to be re-input to the Binder later; this is called incremental binding). The binder accepts library modules as well, linking only those modules required to resolve external references. BND-286 generates a print file displaying a segment map, and error messages.

The Binder is useful for system as well as application programmers. Since application programmers need to develop software independent of any system architecture, the 286 memory protection mechanism is "hidden" from users of the Binder. This allows application tasks to be fully debugged before becoming part of a protected system. (A protected system may be debugged, as well.) System protection features are specified later in the development cycle, using the 286 System Builder. It is possible to link operating system services required by a task using either the Binder or the Builder. This flexibility adds to the ease of use of the 286 utilities.

VAX/VMS resident BND-286 is completely feature compatible with other host versions of BND-286 with the same version number.

VAX*/VMS* RESIDENT 80286 LIBRARIAN

- **Allows Creation and Management of 80286 Object Libraries**
- **Library Functions include Create, Delete, Add, Replace, Copy, Save, Backup and Display**
- **Only Required Modules Linked in When Using Binder or Builder**
- **Compatible with Development System Resident 80286 Librarian**

LIB-286 is the 80286 Librarian. It can be used to create and manage 80286 Object Libraries. By placing often used object modules into libraries, the administrative overhead of managing software modules can be reduced.

VAX/VMS based LIB-286 is completely feature compatible with other host versions of LIB-286 with the same version number.

VAX*/VMS* RESIDENT 80286 MAPPER

- **Flexible Utility to Display Object File Information in Symbolic Form**
- **Compatible with Development System Resident 80286 Mapper**

MAP-286 is a cross reference utility for 80286 object modules. It provides a symbolic listing of the EXTERNAL and PUBLIC symbols in the specified object modules.

VAX/VMS resident MAP-286 is completely feature compatible with development system resident MAP-286 with the same version number.

SPECIFICATIONS

Operating Environment

DEC VAX* 11/780 or compatible model running VMS* operating system V3.4 (or upward compatible versions)

DEC MicroVAX* running Micro VMS (V4.4 or later).

Documentation

Installation guide and user's manuals for the software are supplied with the products.

SUPPORT

Hotline Telephone Support, Software, Performance Report (SPR) Software Updates, Technical Reports and Monthly Newsletters are available.

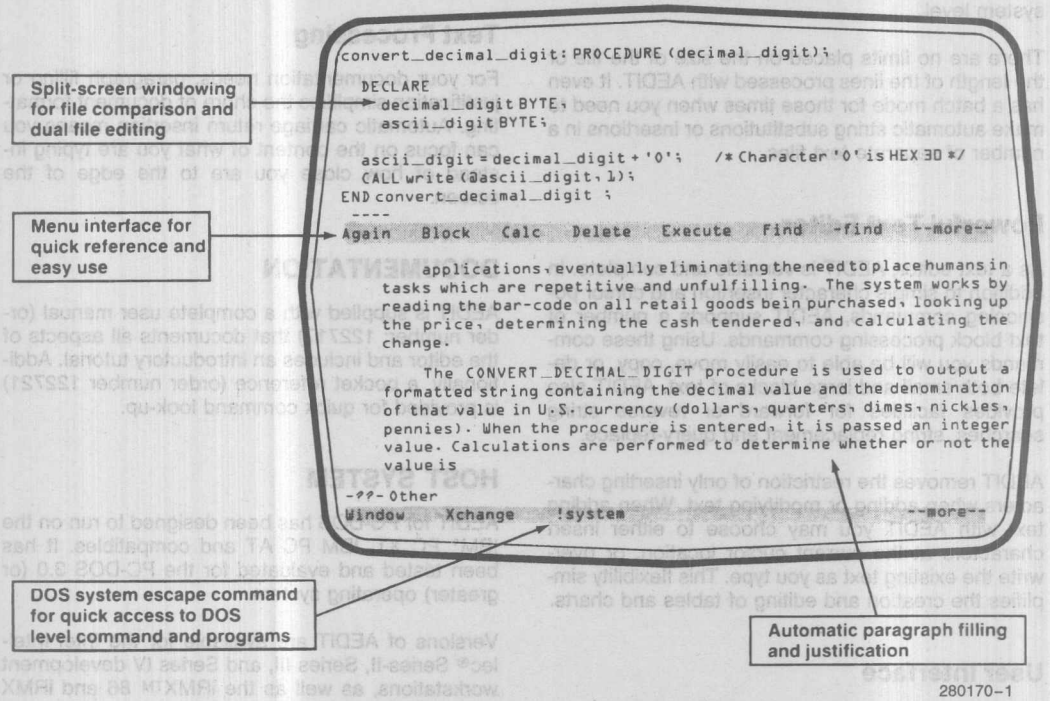
*VAX/VMS are trademarks of Digital Equipment Corporation

ORDERING INFORMATION

Product Code	Description
VVSASM286	ASM-286
	BLD-286
	BND-286
	LIB-286
	MAP-286
MVVSASM286	ASM-286
	BLD-286
	BND-286
	LIB-286
	MAP-286
VVSPLM286	PL/M 286 (VAX)
MVVSPLM286	PL/M 286 (MicroVAX)
VVSC286	C286 (VAX)
MVVSC286	C286 (MicroVAX)

AEDIT
SOURCE CODE AND TEXT EDITOR FOR PC-DOS

- **A Full-Screen Source Code Editing and Documentation Tool Designed Specifically for Software Programmers and Technical Writers**
- **Complete Range of Editing Support—from Document Processing to HEX Code Entry and Modification**
- **Supports System Escape for Quick Execution of PC-DOS System Level Commands**
- **Full Macro Support for Complex or Repetitive Editing Tasks**
- **Dual File Support with Optional Split-Screen Windowing**
- **No Limit to File Size or Line Length**
- **Quick Response with an Easy to Use, Menu-Driven Interface**
- **Configurable and Extensible for Complete Control of the Editing Process**



280170-1

* IBM is a registered trademark of International Business Machines.

October 1986
Order Number: 280170-001

PRODUCT FEATURES

Programmer Support

AEDIT is a full-screen text editing system designed specifically for software engineers and technical writers. With the facilities for automatic program block indentation, HEX display and input, and full macro support, AEDIT is an essential tool for any programming environment. And with AEDIT, the output file is the pure ASCII text (or HEX code) you input—no special characters or proprietary formats.

Dual file editing means you can create source code and its supporting documentation at the same time. Keep your program listing with its errors in the background for easy reference while correcting the source in the foreground. Using the split-screen windowing capability, it is easy to compare two files, or copy text from one to the other. The DOS system-escape command eliminates the need to leave the editor to compile a program, get a directory listing, or execute any other program executable at the DOS system level.

There are no limits placed on the size of the file or the length of the lines processed with AEDIT. It even has a batch mode for those times when you need to make automatic string substitutions or insertions in a number of separate text files.

Powerful Text Editor

As a text editor, AEDIT is versatile and complete. In addition to simple character insertion and cursor positioning commands, AEDIT supports a number of text block processing commands. Using these commands you will be able to easily move, copy, or delete both small and large blocks of text. AEDIT also provides facilities for forward or reverse string searches, string replacement and query-replace.

AEDIT removes the restriction of only inserting characters when adding or modifying text. When adding text with AEDIT you may choose to either insert characters at the current cursor location, or overwrite the existing text as you type. This flexibility simplifies the creation and editing of tables and charts.

User Interface

The menu-driven interface AEDIT provides makes it unnecessary to memorize long lists of commands and their syntax. Instead, a complete list of the commands or options available at any point is always displayed at the bottom of the screen. This makes AEDIT both easy to learn and easy to use.

Full Flexibility

In addition to the standard PC terminal support provided with AEDIT, you are able to configure AEDIT to work with almost any terminal. This, along with user-definable macros and fully adjustable tabs, margins, and case sensitivity, combine to make AEDIT one of the most flexible editors available today.

Macro Support

AEDIT will create macros by simply keeping track of the commands and text you type, "learning" the function the macro is to perform. The editor remembers your actions for later execution, or you may store them in a file to use in a later editing session.

Alternatively, you can design a macro using AEDIT's powerful macro language. Included with the editor is an extensive library of useful macros which you may use or modify to meet your individual editing needs.

Text Processing

For your documentation needs, paragraph filling or justification simplifies the chore of document formatting. Automatic carriage return insertion means you can focus on the content of what you are typing instead of how close you are to the edge of the screen.

DOCUMENTATION

AEDIT is supplied with a complete user manual (order number: 122717) that documents all aspects of the editor and includes an introductory tutorial. Additionally, a pocket reference (order number 122721) is provided for quick command look-up.

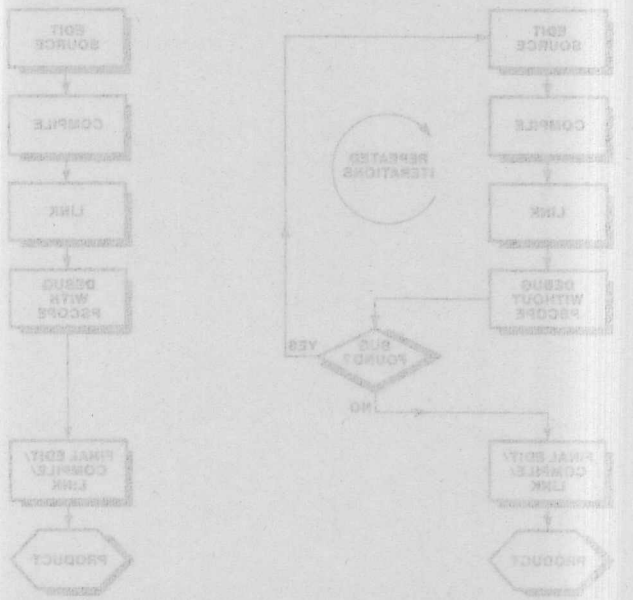
HOST SYSTEM

AEDIT for PC-DOS has been designed to run on the IBM* PC XT, IBM PC AT and compatibles. It has been tested and evaluated for the PC-DOS 3.0 (or greater) operating system.

Versions of AEDIT are available for the Intel Intellec® Series-II, Series III, and Series IV development workstations, as well as the iRMX™ 86 and iRMX 286 operating systems.

ORDERING INFORMATION

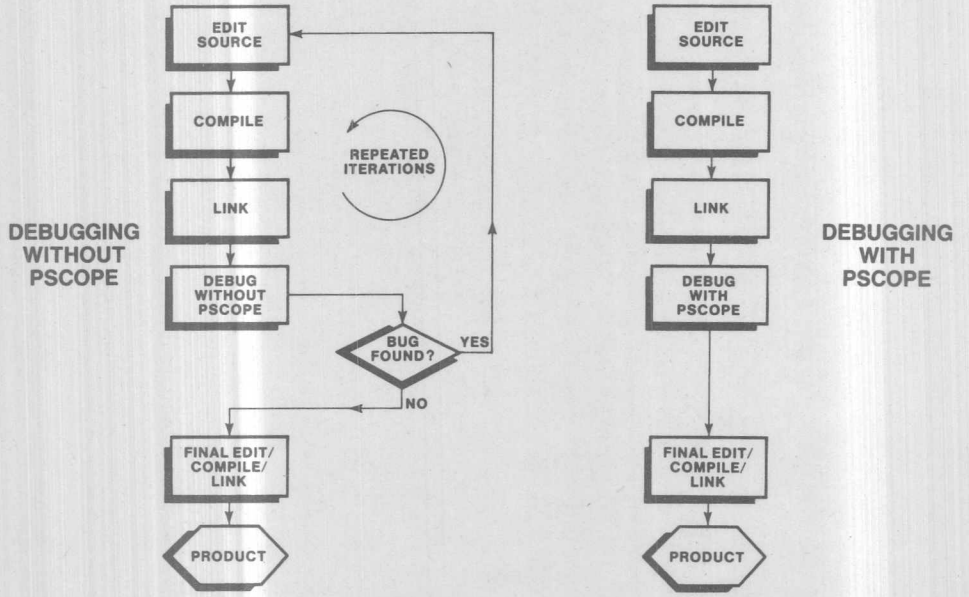
Order Code	Description
D86EDIEU	AEDIT Source Code Editor Release 2.2 for PC-DOS with supporting documentation
122717	AEDIT-DOS User's Guide
122721	AEDIT-DOS Pocket Reference



PSCOPE
HIGH-LEVEL PROGRAM DEBUGGER FOR iRMX™, XENIX*, SERIES III AND SERIES IV

- **Provides Source Level Debugging Capabilities for High-Level Languages and Assembly-Level Languages**
- **Permits Creation of High-Level Program Patches using PSCOPE High-Level Language Constructs**
- **Sets Breakpoints and Traces Program Execution**
- **On-Line Help Facilities**
- **Code Disassembly/Assembly and Assembly-Level Patching**
- **Symbolic Debugging Capabilities**
  - **— Maintains Type Information about Variables**
  - **— Supports Symbolic Access to Dynamic Local Variables**
  - **— Maintains a Virtual Symbol Table for Program Variables**
  - **— Allows Definition of User-Defined Debugging Variables and Procedures**
  - **— Accesses Memory Locations and Program Variables using Program-Defined Names**
- **Single Stepping**

PSCOPE is an interactive, symbolic debugger for high-level-language programs written in PL/M, Pascal, and FORTRAN. The iRMX™ PSCOPE and XENIX™ PSCOPE products support debug of programs written in assembly language. XENIX PSCOPE provides additional C language support and can use XENIX operating system commands without leaving or altering the debug environment.



280266–1

280266–2

*XENIX is a trademark of Microsoft Corporation.

October 1987
Order Number: 280266-002

PSCOPE OVERVIEW

With PSCOPE, a user can load an application program into host system memory, set breakpoints at symbolic or numeric addresses, trace program execution, and create patches. Other debugging aids include the ability to single-step a program through high-level-language statements or procedures, to display and modify program variables, to inspect files, and to personalize the debugging environment.

MAJOR FEATURES

The following sections describe the major features of the PSCOPE high-level debugger.

Symbolic Debugging

With symbolic debugging, a user can examine or modify a memory location by using its symbolic reference. A symbolic reference is a procedure name, variable name, line number, or program label corresponding to a location in the user program's memory space. For example, to display the value of the program variable *linesend*, users need only GO TIL the variable is active, and type the variable's name. Note that * is the PSCOPE prompt.

```
*linesend
50
```

Notice that PSCOPE returns the variable value without the user having to indicate the variable's type. The capability to recognize a variable's type and scope is a special feature of PSCOPE's support of symbolics. Few other debuggers offer this feature.

Consider another example. Suppose the user's program has an array of employee records called *emprec* that includes salary and other employee information. Using PL/M, the user might declare it as follows:

```
DECLARE emprec (100) STRUCTURE
                     (name (20) BYTE,
                      ss (10) BYTE,
                      number INTEGER,
                      salary REAL);
```

With PSCOPE, to determine the salary of the nth employee, the user need only type:

```
emprec[n].salary
```

PSCOPE would then respond:

```
2.200E + 03
```

Patch

A patch is a set of PSCOPE high-level commands that augments or replaces a section of the user's program. With patching, the user can modify a program's algorithm and verify the effect of modifications without having to edit source, recompile and relink.

For example, the following patch indicates that if the value of the variable *x* is 0, then the current execution point ($ is a PSCOPE pseudo-variable for the current execution point) will be line #39. If *x* is not 0, the value of *y* will be set to the value of *linesend* minus the value of *x*. The execution point will not change.

```
*DEFINE PATCH #37 TIL #39 = DO
.*IF x = 0 THEN $ = #39
.*ELSE
.*y = linelength - x
.*ENDIF
*END
*
```

A patch can also be used to bypass statements. The following command causes lines #13 through #15 to be skipped, resuming execution at line #16.

```
*DEFINE PATCH #13 TIL #16 = nop
```

Breakpoints

Breakpoints suspend program execution at specified locations. The user can then enter PSCOPE commands, construct patches, examine or change program variables and registers. PSCOPE for the XENIX and iRMX operating systems also allow memory disassembly. Execution can be resumed from the breakpoint or from any other point.

A breakpoint specification is the address where program execution stops. The address can be specified as a symbolic address, segment/offset pair, or as a high-level-language statement number.

For example:

```
*GO TIL: count_lines
 [Break at count_lines]
```

Debug Procedures

A debug procedure is a group of PSCOPE commands that are invoked by a name. Debug procedures can be saved and then recalled for use in later debug sessions.

Following is the definition of a debug procedure called *sum*. It consists of a DO command block that returns the sum of all the parameters passed to it. It contains two local variables, *n* and *i*.

```
*DEFINE PROC sum = DO
.*DEFINE LONGINT n = 0
 .*DEFINE INTEGER i = 0
 .*COUNT % np
 ..*n = n + %(i)
 ..*i = i + 1
 ..*ENDCOUNT
 .*RETURN n
 .*END
 *
```

To execute the debug procedure, invoke it by name as follows:

```
*sum(2,3,5)    /*Executing the debug procedure*/
```

The benefits of debug procedures include simplification of command invocation for groups of commands and automation of the software verification process. A procedure can be defined to iteratively generate test values, execute the program with new input values, and record results. Thus, debug procedures can be used to develop and run comprehensive "batched" tests.

Break Registers

A break register is a named set of one or more breakpoint specifications. After defining the contents of break registers, the user can execute a program using the specifications in one or more of these break registers. For example, here is the definition of a break register called *break1*:

```
*DEFINE BRKREG break1 = error_check
```

To execute and break just before procedure error_check, specify use of break register *break1* in the GO command.

```
*GO USING break1
```

Break registers are useful for storing sets of specifications to be recalled in later debug sessions. They also enable users to call a previously defined debugging procedure when a specification in a break register is met. This gives the user a powerful tool for creating conditional constructs to observe program behavior. For example, assume a user wants to break at three breakpoints, *term, value,* and at line #68. Before breaking at the procedures *term*, and *value*, the user wants to execute a debug procedure. The following example first defines the procedure (pr1) and then defines the break register (break2) that calls the procedure.

```
*DEFINE PROC prt1 = DO
 .*IF x > 0 THEN RETURN true
 .*ELSE RETURN false
 .*ENDIF
 .*END
 *
*DEFINE BRKREG break2 = (term, value) call
 prt1, #68
```

To run the program using this break register, enter:

```
*GO USING break2
```

Trace Registers

The PSCOPE trace feature displays a trace message when the program it is executing reaches a specified address. The trace message identifies the current execution point, but no break occurs.

The following example defines a trace register named *trace1* that contains tracepoints at statements #80, #124, and at the procedure *error* in the current module:

```
*DEFINE TRCREG trace1 = #80, #124, error
```

A GO command using *trace1* displays a message each time statements #80, #124, or procedure *error* is executed.

On-Line Help

PSCOPE provides on-line help. In addition to obtaining help on topics from a help list, extended versions of PSCOPE error messages can be displayed.

Stepping

With PSCOPE commands users can single-step through high-level-language statements (the LSTEP command), and procedures (the PSTEP command). The LSTEP and PSTEP commands display the statement number of the next high-level-language statement. For example:

```
*LSTEP
[Step at :PAGER#42]
```

The iRMX and XENIX PSCOPE debuggers also have the capability to step through machine instructions (the ISTEP command).

Literally Definitions

LITERALLY definitions are shorthand names for previously defined character strings. They give the user the convenience of being able to customize the debug environment. For example:

```
*DEFINE LITERALLY lit = 'literally'
*DEFINE lit def = 'define'
*def lit stacktop = 'word ss:sp'
```

The Editor

PSCOPE includes an internal editor which provides a subset of Intel's AEDIT text editor's features. With this editor, users can create and modify debug constructs such as patches, debug procedures, and LITERALLY definitions. The internal editor can be used to view source program files on the screen.

THE PSCOPE COMMAND LANGUAGE

The PSCOPE commands are versatile and powerful, yet easy to learn and use. With them, the user can build a high-level environment in which to examine and modify execution of the program under development.

The PSCOPE commands can be divided into functional categories.

Emulation commands instruct PSCOPE-86 to execute the user program. They consist of GO and the two stepping commands, LSTEP, and PSTEP.

Debug environment commands define PATCHes, debug PROCedures, debug *variables*, LITERALLYs, break registers (BRKREG), and trace registers (TRCREG) using the DEFINE command. A user can delete these definitions with the REMOVE command.

Block commands consist of DO-END, COUNT-END, REPEAT-END, and IF-THEN-ELSE constructs. They can be used alone or within debugging procedures and patches.

String functions concatenate strings (CONCAT), return the string length (STRLEN), return a substring (SUBSTR), and accept console input (CI).

Utility commands are general-purpose commands for use in a debugging environment. They consist of the following:

$	is a pseudo-variable that represents the current execution point.
ACTIVE	is a function that determines whether a specified dynamic variable is currently defined on the stack or not.
BASE	sets or displays the current radix.
CALLSTACK	displays the dynamic calling sequence stored on the stack.
DIR	displays all objects of a specified type.
EDIT	invokes the internal, menu-driven, text editor.
EVAL	returns the value of a symbol in binary, decimal, hexadecimal, and ASCII.
EXIT	returns control to the host operating system.
HELP	provides on-line help for selected topics and error messages.
NAMESCOPE	This pseudo-variable represents the current scope of a variable. Gives access to variables without need to use the fully qualified symbolic reference.
OFFSET$OF	is a function that returns the offset of a specified address (virtual or symbolic).
SELECTOR$OF	is a function that returns the selector of a specified address (virtual or symbolic).
WRITE	writes variables and strings to the console's screen.

File handling commands access disk files. The user can load program files to be debugged (LOAD), save patches, debugging procedures, debugging variables, LITERALLYs, and debugging registers in a disk file (PUT and APPEND), read-in these definitions during later debugging sessions (INCLUDE), and record a debugging session in a disk file for later analysis (LIST and NOLIST).

Register access commands provide access to the microprocessor registers and flags. The REGS command displays the registers and the set flags. Users can also inspect or change an individual register by specifying its mnemonic. FLAG represents the flag word. The user can inspect or change each flag separately as a Boolean variable.

ADVANCED PSCOPE COMMANDS FOR iRMX™ AND XENIX OPERATING SYSTEMS

In addition to the basic PSCOPE capabilities, iRMX and XENIX PSCOPE offer complete debugging support for assembly language programs. This support includes the ISTEP single step command, an assembler/disassembler, and register examination and modification for both the main processor and the math coprocessor as follows:

XENIX	80286/80287
iRMX	8086/8087

XENIX PSCOPE also supports debugging of C language programs. Another feature of XENIX PSCOPE allows the user to invoke any XENIX operating system command and return to debugging without altering the state of the debug session.

The Disassembler and Single-Line Assembler

With the disassembler, memory can be displayed as assembly language mnemonics. The next example shows how the ASM command displays the first assembly language instruction that makes up the high-level-language statement #26.

```
*ASM #26
3348H:00CCH FF36000000 PUSH WORD PTR 000H
3348H:00D0H B020       MOV AL,20H
```

An instruction can be changed using the single-line assembler as follows:

```
*ASM 3348H:00D0H = 'MOV AL, 25H'
```

SAMPLE DEBUG SESSION

The following sample debug session illustrates some of PSCOPE's capabilities. The example program (Figure 1) is written in Pascal. To utilize PSCOPE's symbolic debugging capabilities, the program was compiled with the debug option. After the program is linked and bound (bind option), the executable code can be debugged on PSCOPE.

The following example of the debugging session demonstrates the use of the LITERALLY command to personalize the debug environment. This example assumes PSCOPE has been invoked and the PSCOPE prompt (*) is present on the screen. PSCOPE keywords are in uppercase for the examples, but case is not significant in the PSCOPE command language. No devices or directories will be included with file access commands.

```
*DEFINE LITERALLY d = 'DEFINE'
*d LITERALLY l = 'LITERALLY'
*d l br = 'BRKREG'
*d l tr = 'TRCREG'
```

Next the load time locatable code is loaded, and a directory of the module is requested. In the directory, program symbols are listed with their types. The types corresond to PSCOPE memory object types.

```
*LOAD maxmin.86
*DIR
DIR of:CALC
PQ_OUTPUT ..... TEXT (file)
PA_INPUT ..... TEXT (file)
B ..... integer
A ..... integer
SUM ..... procedure
  X ..... integer
  Y ..... integer
  Z ..... integer
DIFFERENCE ..... procedure
  X ..... integer
  Y ..... integer
  Z ..... integer
MAXMIN ..... procedure
  X ..... integer
  Y ..... integer
```

The next example illustrates using the PSTEP stepping command. PSTEP executes procedures as a single step. When the program requests input, the user must enter values for the stepping to continue. PSCOPE responds with a break message inside brackets ([ ]).

```
*PSTEP
[Step at :CALC#21]
*PSTEP
        Input two integers
[Step at :CALC#22]
                          19 ← user input
                           4 ← user input
[Step at :CALC#23]
*PSTEP
        The sum is 76          ← note
*PSTEP
[Step at :CALC#24]
        The difference is 15
[Step at :CALC#24]
*PSTEP
        The maximum is   19
        The minimum is    4
```

```
Source File: MAXIN.PAS
Object File: MAXIN.OBJ
Controls Specified: DEBUG
                    (* This program reads two integers and  *)
                    (* determines which is greater.         *)
 STMT  LINE  NESTING  SOURCE TEXT: MAXMIN.PAS
    1     1        0        0  program calc(input,output);
    2     2        0        0  var a,b:integer;

    3     4        0        0  procedure sum(s,y:integer);
    4     5        1        0  var z:integer;
    5     6        1        0  begin
    5     7        1        0    z: = x*y;
    6     8        1        1    writeln('The sum is',z);
    7     9        1        1  end;

    8    11        0        0  procedure difference(s,y:integer);
    9    12        0           var z:integer;
   10    13        1        0  begin
   10    14        1        0    z: = abs(x-y);
   11    15        1        1    writeln(''The difference is',z);
   12    16        1        1  end;

   13    18        0        0  procedure maximum(x,y:integer);
   14    19        1        0  begin
   14    20        1        1    if x < y then writeln ('The maximum is',y),
                                                The minimum is',x);
   16    21        1        1    if y < x then writeln ('The maximum is',x,
                                                The minimum is',y);
   18    22        1        1    if x = y then writeln ('The inputs are equivalent')
   20    23        1        1  end;

   21    25        0        0  begin
   21    26        0        1    repeat (*forever*)
   21    27        0        2      write('Input two integers');
   22    28        0        2      readln(a,b);
   23    30        0        2      sum(a,b);
   24    31        0        2      difference(a,b);
   25    32        0        2      maximum(a,b);
   26    33        0        2    until 1 < 0
   27    34        0        2  end.
```

Figure 1. Sample Program

There is a bug in the program. The procedure sum should add the input values, 19 and 4. Instead, it multiplies them. The code on line #5 should specify x+y instead of x*y. With PSCOPE, it is easy to patch this line of code and immediately execute the program without recompiling and relinking.

```
*DEFINE PATCH #5 TIL #6 = z = x+y
```

The next example uses the GO TIL command to illustrate that the patch works.

```
*TO TIL #21
                Input two integers      4
                                       19
                The sum is             23
                The difference is      15
                The maximum is         19
                The minimum is          4
[Break at #21]
```

The final example illustrates the use of a PSCOPE PROC (debug procedure). The PROC is named pr1. Then a BRKREG (break register) is defined to call the procedure. Notice the use of the LITERALLY definitions in the examples.

```
*d PROC pr1 = DO
.*WRITE 'numbers and product are: ',a,b,a,*b
.*WRITE USING ('0,>') 'break?'
.*IF CI = = 'y' THEN RETURN true
..*             ELSE RETURN false
..*ENDIF
.*END
*d br b3 = #21 CALL pr1
*GO USING b3
                Input two integers      23
                                        24
                The sum is              47
                The difference is        1
                The maximum is          24
                The minimum is          23

numbers and the product are: +23 +24 +552
break ? y
[Break at #21]
```

When a debugging session is complete, exit PSCOPE as follows. The debugger will close any open files, write a message to the screen and return to the operating system.

```
*EXIT
PSCOPE terminated
```

BENEFITS

As an interactive, symbolic, high-level language debugger, PSCOPE brings to debugging the same type of productivity enhancements that high-level languages bring to writing software. PSCOPE's benefits are listed below:

- A shortened development cycle. Breakpoints, tracing, and patching decrease the number of edit/compile/link iterations.
- Improved debugging productivity. Since PSCOPE language constructs enable the use of high-level functions and procedures, symbolics and data structures, improvement in debugging productivity is analogous to programming in high-level languages.
- Increased software reliability. Debugging procedures can automate the software testing process.
- Improved project management. Software engineers can debug modules separately. Procedures can be substituted for program stubs.

SPECIFICATIONS

PSCOPE for the iRMX™ Operating System

(for iRMX operating system release 5 or greater on Systems 86/310, 86/330A, and 80/380)

Languages: •PL/M-86 •Pascal-86 •FORTRAN-86 •ASM86

Documentation: *PSCOPE-86 High-Level Program Debugger User's Guide* (for iRMX Operating Systems) Order number: 165496

Memory: 110K bytes for iRMX PSCOPE-86 software and buffers

Order Code	Description
iPSC 86 RMX	PSCOPE Program Debugger for the iRMX Operating System

PSCOPE for the XENIX Operating System

(PSCOPE requires an 80286-based system running the Intel XENIX 286 operating system.)

Languages: Supports any Intel or ISV-supplied 80286 language generating Intel 286 object module format load-time locatable modules. These include:

•PL/M •C •FORTRAN
•Pascal •ASM

Documentation: *PSCOPE 286 User's Guide for XENIX Systems,* Order Number: 122281. *PSCOPE 286 Pocket Reference,* Order Number: 122282.

Memory: 286 bytes for XENIX PSCOPE software and buffers.

Order Code	Description
iPSC286XNXSU	Single-user license
iPSC286XNXRO	Incorporation license
iPSC286XNXRF	Royalty fees for incorporations
iPSC286XNXBY	Buy-out license

PSCOPE for the Series III/Series IV Operating System

(PSCOPE-86 runs on an Intellec® system, either stand-alone or in an NDS-II network configuration.)

Languages: •PL/M 86/88 •Pascal 86/88
•FORTRAN 86/88

Documentation: *PSCOPE-86 High-Level Program Debugger User's Guide,* Order number 121790

Memory: The system configuration must include 512K bytes of application memory space. The debugger requires 96K bytes of memory.

Order Code	Description
iMDX-333	for Series III and Series IV
III-951A	I2ICE software for Series III 8″ single density disk drive
III-951B	I2ICE software for Series III 8″ double density disk drive
III-951C	I2ICE for Series IV with 5 1/4″ double density disk drive

An enhanced version of PSCOPE with source-code display is available for the PC-DOS operating system. See Data Sheet order number 280194 for information on PSCOPE-86 for DOS operating systems.

PSCOPE-86 FOR DOS
HIGH-LEVEL APPLICATION PROGRAM DEBUGGER

- **Debugs PL/M-86, Pascal-86, iC-86, FORTRAN-86, and ASM86 Programs**
- **Displays Program Text on the Screen During Debugging:**
  - **— Uses the Listing File to Display Program Text**
  - **— Displays Source Code on Program Step, at Execution Break Points, or on User Request**
- **Disassembles Memory and Provides an Interactive Assembler**
- **Permits Creation of Program Patches Using High-level Language Constructs**
- **Supports Access to DOS Operating System Commands**
- **Offers Symbolic Debugging Capabilities:**
  - **— Supports Access to Memory by Program Defined Variable and Program Names**
  - **— Maintains Type Information About Variables**
  - **— Allows Definition of User-defined Debugging Variables and Procedures**
- **Single-steps Through Assembly Language Instructions, High-level Language Statements, or Procedures**
- **Sets Break Points and Traces Program Execution**
- **Runs Under the PC-DOS Version 3.0 or Greater**

PSCOPE-86 for DOS is an interactive, symbolic debugger for high-level language programs written in iC-86, PL/M-86, Pascal-86, and FORTRAN-86, and for assembly language programs written in ASM86. PSCOPE-86 for DOS runs under the PC-DOS operating system, version 3.0 or greater.

```
*LIST a:debug.log
*LOAD \progdir\leapyr .86
*SET :leapyr to \listdir\leapyr .lst lang pascal
*DIR LINE
DIR of :LEAPYR
#1     #5    #6    #7    #8    #9    #10   #11   #12   #13
#14    #15   #16   #17   #18   #21   #22   #23   #25
*PRESRC=0;POSTCRC=0;SOURCE=true;GO TIL #13
Enter the number of a month.
2
Enter any year, like 1985.
1984
[Break at :LEAPYR#13]
=>   13   24   0   2          CASE month of
+LSTEP
[Step at LEAPYR#16]
=>   16   27   0   3        2:(* leap year *)
           IF (year mod 4 = 0) AND ((year mod 100 <> 0) OR
                                    (year mod 400 = 0))
*
```

280194-1

October 1986
Order Number: 280194-002

MAJOR FEATURES

With PSCOPE-86 for DOS, a user can load an application program, set break points at symbolic or numeric addresses, trace program execution, and view source code text. Program bugs can be patched using high-level PSCOPE commands or assembly code. The corrections can be tested without leaving the PSCOPE software.

Other debugging aids include the ability to single-step a program through assembly language instructions, high-level-language statements, or procedures, to display and modify program variables, to inspect files, and to personalize the debugging environment.

The following sections describe some of the major features of PSCOPE-86 for DOS.

Source Display

With the DOS version of PSCOPE-86, a user can correlate a module under debug to a source code file. Then, when break points are encountered, source text is displayed along with the break message and line number of the break point. The number of source lines displayed before and after a break point can also be defined by the user.

View all or part of the listing file on command. The following example uses the PSCOPE command to list the current module. The asterisk (*) is the PSCOPE prompt, the command follows, and after pressing <Enter>, PSCOPE responds with a list file.

```
*SHOWSRC #1 LENGTH 28
 1    1    0    0  program leapyr (input,output);
                     (* Input month and year, receive number of days *)
 2    5    0    0  var year    :integer;
 3    6    0    0      month   :integer;
 4    7    0    0      nrdays  :integer;

 5    9    0    0  begin

 5   11    0    1     month := 0;
 6   12    0    1      year := 0;
 7   13    0    1    nrdays := 0;

 8   15    0    1  writeln('Enter the number of a month.');
 9   16    0    1  readln(month);
10   17    0    1    while month <> 999 do

11   19    0    1    begin
11   21    0    2  writeln('Enter any year, like 1985.');
12   22    0    2  readln(year);

13   24    0    2     CASE month of
14   25    0    3                    4,6,9,11:nrdays := 30;
15   26    0    3           1,3,5,7,8,10,12:nrdays := 31;
16   27    0    3                          2: (* leap year *)
                      IF (year mod 4 = 0) AND ((year mod 100 <> 0) OR
                                               (year mod 400 = 0))
                                     THEN nrdays := 29
17   31    0    3                    ELSE nrdays := 28;

19   33    0    3      end;

21   35    0    2    writeln('Number of days in the month is',nrdays);

22   37    0    2    writeln('Enter the number of a month.');
23   38    0    2    readln(month)
                    end;
25   40    0    1  end.
```

Single-Stepping

PSCOPE has two commands to single-step through high level instructions and display source code. The commands differ in how they handle program calls. The following example illustrates the LSTEP command.

```
*LSTEP
[Step at :LEAPYR#17]
=>   17  31  0  3                    ELSE nrdays := 28;
     19  33  0  3                end;
*LSTEP
[Step at :LEAPYR#21]
=>   21  35  0  2          writeln('The number of days in the month is',nrdays);
```

PSCOPE can single-step through code at assembly level and display assembly mnemonics as in the following example which uses the ISTEP command.

```
*ISTEP
:LEAPYR
512A:00FEH        C70600000000          MOV WORD PTR 0000H,0
```

Symbolic Debugging

With symbolic debugging, a user can examine or modify a memory location by using its symbolic reference. A symbolic reference is a procedure name, variable name, line number, or program label that corresponds to a location in the user program's memory space. For example, to display the value of the program variables, users need only execute the program until the variable is active and type that variable's name.

```
*LSTEP
[Step at :LEAPYR#22]
=>   22  37  0  2          writeln('Enter the number of a month.');
*month
+2
*year
+1900
*nrdays
+28
```

Define the Debug Environment

With the PSCOPE high-level program debugger, a user can define the debugging environment within PSCOPE software. You can define break points and trace points. With PSCOPE, you can write macros that set the debug environment when PSCOPE is invoked, or these macros can be included at any time during the debugging session. Shorten commands with literal definitions, try program bug fixes with patches and procedures, or write procedures to control program execution. All debug variables and procedures can be saved in files and reused.

BREAK REGISTERS AND TRACE REGISTERS

Breaks occur at addresses in the program under execution. The user can enter physical addresses or symbolic addresses to halt program execution. With PSCOPE, you can easily break at executable statement addresses by using line numbers. Simply use the PSCOPE directory command with the line option (DIR LINE) to get a directory of line numbers. Then define a break register or a trace register to stop at these addresses.

A break register (BRKREG) stops program execution and returns a PSCOPE prompt (*). A trace register (TRCREG) displays a message and continues program execution. Following are examples of how to define a break register and a trace register.

```
*DEFINE BRKREG stop = #22
*DEFINE TRCREG stop2 = #17
```

DEBUGGING PROCEDURES

Debugging procedures are groups of PSCOPE commands that have been labeled. Writing procedures with PSCOPE commands is much the same as writing high-level language procedures. A procedure can be used for any definable function during a debugging session, and it can be used with a program under execution.

In the LEAPYR program, the while loop continues until 999 is entered for a month number. The following example of a PSCOPE procedure (PROC) that querys the user about halting execution. If the answer is yes ('Y' or 'y'), the procedure sets *month* to 999.

```
*DEFINE PROC query = DO
.*WRITE USING ('Do you want to quit? Enter Y for yes.')
.*DEFINE CHAR ccc = CI
.*WRITE ccc
.*IF ccc == 'Y' or 'y' then
..*month == 999
..*RETURN = true
..*else RETURN = false
.*endif
.*END
*
```

To call this procedure while the program is executing, define a break register and use it with the GO command as follows:

```
*DEFINE BRKREG stop = #22 CALL query
*GO USING stop
```

PSCOPE PATCHES

A PSCOPE patch is used to temporarily correct run-time errors in the program under debug. A patch can be an additional line (or lines) in a program, or can be used to replace lines in a program. PSCOPE enables both high-level patches (the PATCH command) and assembly-level patches (the ASM command).

High-Level Patch

In the LEAPYR program, the way to exit the program is to enter 999 for the month. However, nothing instructs the user to do this. With a high-level patch, it is simple to add a line of code to the program. Following is an example.

```
*DEFINE PATCH #22 = WRITE 'To exit the program, enter 999.'
```

When the program is executed, the patch is used automatically. There is no need for a break register. Program execution stops at line number 22, the patch message is displayed, and program execution continues at line number 22. It is also possible to replace lines by using the TIL option in a high-level patch. Then program execution continues from the line number, or address, defined after the TIL. To simply eliminate lines of code, set the line to NOP as follows:

```
*DEFINE PATCH #18 = NOP
```

Assembly-Level Patch

Assume there is a typo in the LEAPYR program. Instead of the else condition setting nrdays to 28, it sets nrdays to 29, making every year leap year. Use the ASM command first to display assembly code as in the following example.

```
*ASM #17 LENGTH 4
:LEAPYR
521A:01E0H    C70600001D00      MOV  WORD PTR 0000H,001DH  ;+29T
521A:01E6H    EB06              JMP   $+0008H              ; A=01EEH
=>  17  31  0 3                   ELSE nrdays := 29;
521A:01E8H     C70600001D00     MOV  WORD PTR 0000H,001DH  ;+29T
521A:01EEH EB00 JMP $+0002H          ; A=01F0H
```

Notice that source code can be displayed to assist you in finding the ELSE statement. However, source display can be eliminated simply by setting the variable SOURCE to false. After finding the address for the correct line of code, use the ASM command to change the second 29 to 28. Notice in the following example, 'word' is sufficient for the assembly mnemonic. The 'ptr' mnemonic is unnecessary.

```
*ASM 521A:01E8H = 'mov word 0000H,001Ch'
521A:01E8H    C70600001C00      MOV  WORD 0000H,001DH
```

LITERALLY DEFINITIONS

LITERALLY definitions are shortened names for previously defined character strings. LITERALLY definitions save keystrokes or improve clarity. For example, the following LITERALLY definition replaces the command DEFINE with the abbreviation DEF.

```
*DEFINE LITERALLY def = 'DEFINE'
```

Save and Restore the Debug Environment

All debug variables and procedures can be saved in a file for future debug sessions. To save everything in a file, use the PUT command as follows:

```
*PUT a:debug.mac DEBUG
```

The saved file can be used as a macro and invoked automatically with PSCOPE by using the following invocation command to start PSCOPE.

```
C:>PSCOPE MACRO(a:debug.mac)
```

After PSCOPE is loaded, a list of all the commands in the macro will print to the screen and will be included in the debug environment. It is also possible to include a macro after PSCOPE is loaded. The following example uses the NOLIST option to prevent the commands from writing to the screen.

```
*INCLUDE a:debug.mac NOLIST
```

The Internal Editor

PSCOPE has an internal editor that is a version of Intel's Aedit. Use this editor to correct source code as program fixes are confirmed with PSCOPE. The editor can also be used to create macros, procedures, or correct command lines.

Escape to DOS

PSCOPE has an escape function to enable access to the DOS operating system commands. This is very useful to verify a file location or print a file. Any DOS operating system command is accepted after entering the 'bang', explanation point, (!). The following is an example of the ESCAPE command.

```
*!print a:debug.mac
```

The DOS print message will appear on the screen, and then the PSCOPE prompt. Once the printing is complete, you are again in PSCOPE withoutaltering the debug environment.

The PSCOPE Command Language

The syntax of PSCOPE commands resembles that of a high-level language. The PSCOPE command language is versatile and powerful while remaining easy to learn and use because commands are often self explanatory like GO. GO starts execution of the user program.

The PSCOPE command language can be divided into functional categories.

- Emulation commands instruct PSCOPE to execute the user program. They consist of GO and the three stepping commands, ISTEP, LSTEP, and PSTEP.
- Debugging environment commands define PATCHes, debugging PROCedures, debugging *variables*, LITERALLYs, break registers (BRKREG), and trace registers (TRCREG) using the DEFINE command. A user can also delete these definitions with the REMOVE command.
- Block commands consist of DO-END, COUNT-END, REPEAT-END, and IF-THEN-ELSE constructs. They can be used alone or within debugging procedures and patches.
- String functions concatenate strings (CONCAT), return the string length (STRLEN), return a substring (SUBSTR), and accept console input (CI).
- Utility commands are general-purpose commands for use in a debugging environment. They consist of the following:

!	accesses the DOS operating system commands.
$	is a pseudo-variable that represents the current execution point.
ACTIVE	is a function that determines whether a specified dynamic variable is currently defined on the stack.
ASM	assembles or disassembles memory.
BASE	sets or displays the current radix.
CALLSTACK	displays the dynamic calling sequence stored on the stack.
DIR	displays all objects of a specified type.
EDIT	invokes the internal, menu-driven text editor.
EVAL	returns the value of a symbol in binary, decimal, hexadecimal, and ASCII.
EXIT	returns control to the host operating system.
HELP	provides on-line help for selected topics and selected error messages.
NAMESCOPE	is pseudo-variable that represents the current scope of a variable. It gives access to variables without requiring a fully qualified symbolic reference.
OFFSET$OF	is a function that returns the offset of a specified address (virtual or symbolic).
SELECTOR$OF	is a function that returns the selector of a specified address (virtual or symbolic).
WRITE	writes variables and strings to the console's screen.

- File handling commands access disk files. The user can load program files to be debugged (LOAD), save patches, debugging procedures, debugging variables, LITERALLYs, and debugging registers in a disk file (PUT and APPEND), read-in these definitions during later debugging sessions (INCLUDE), and record a debugging session in a disk file for later analysis (LIST and NOLIST).
- Register access commands provide access to the 8086/8088 registers and flags.

  The REGS command displays the 8086/8088 registers and flags. Users can also inspect or change an individual register by specifying its mnemonic. For example, CS represents the code segment register.

  The FLAG pseudo-variable represents the 8086/8088 flag word. The user can also inspect or change each flag separately as a Boolean variable. (For example, TFL represents the trap flag).

  PSCOPE provides register access for programs that perform real arithmetic. There is a built-in 8087 math coprocessor emulator, or there is a CH8087 option with the LOAD command to tell PSCOPE to access the hardware (8087 math coprocessor chip) registers. Access or change the 8087 registers by name.
- Source display commands are used to view a specified number of lines of source text at break points or on demand. LPATH or SET directs PSCOPE to the source text file. SOURCE is the pseudo-variable used to determine if source text will be displayed at break points. With PRESRC and POSTSRC, the user can determine how many lines of source code will be displayed before and after the line at the break point. The SHOWSRC command enables the display of source code outside of program execution.

SPECIFICATIONS

Memory Requirements

PSCOPE-86 for DOS requires approximately 300KB of memory for PSCOPE software and buffers.

DOS Version

PSCOPE is designed to run on the DOS operating system version 3.0 or greater.

Language Support

iC-86

PL/M-86

FORTRAN-86

ASM86

PASCAL-86

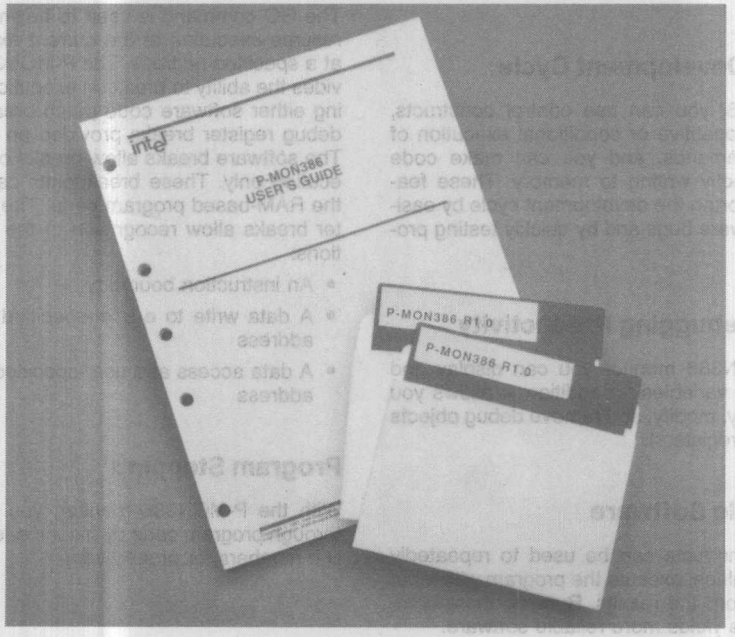
ORDERING INFORMATION

Order Code	Description
D 86 PSC 86	High-Level Software Debugger

- **Program Execution Control, Including Breakpoints and Single-Step Execution through Assembly Level Instructions in Both Real and Protected Mode**
- **Examine and Modify Memory, I/O Ports, and Processor Registers**
- **Examine and Modify Descriptor Tables and the Task State Segment**
- **Download 8086, 80286, and 80386 Code in Intel Object Module Format**
- **Full Symbolic Support for Intel OMFs**
- **Disassemble Memory in 386 Instruction Mnemonic Form**
- **Host Software Executable on the Intel System 286/310 with Intel XENIX* Release 3.0, Update 3 or on IBM PC AT running DOS 3.1**
- **User-Friendly Human-Interface Provides Command Line Editing, On-Line Syntax Guide, and a Command History**
- **Configurable for Variety of Target Systems**

The Intel PSCOPE Monitor 386 (P-MON386) is a debug monitor for 80386-based systems, and is designed to provide software development aid for systems programmers. It can access and control all of the 80386 visible user-hardware resources without any assistance from an operating system. With the help of this monitor, a user can download a program into the target prototype memory, set hardware and software breakpoints, examine/modify memory and processor registers, and control program execution. This monitor is supplied with a serial driver for the 8251 and the 8254, but it can be configured to run on any 80386-based target board with a user-supplied communication driver.

The P-MON386 package includes the diskettes that contain the host software to be loaded into a host, the diskettes that contain the target software to be loaded into the 80386-based target board, and the *PSCOPE Monitor 386 User's Guide.*



280198-1

*XENIX is a trademark of Microsoft Corporation.

October 1987
Order Number: 280198-003

OVERVIEW

P-MON386 provides design aid and debug support for Intel customers who are designing software to run on the 80386 high performance microprocessor with integrated memory management. P-MON386 is hosted on the Intel System 286/310 with XENIX release 3.0, with Update 4, or IBM PC AT with Intel Above Board and running DOS 3.1 or higher and allows the user to perform the following tasks:

- Download Intel 8086, 80286, and 80386 object module formats
- Examine/modify memory, I/O ports, processor registers, descriptor tables, and the task state segment
- Convert addresses from virtual to linear, linear to physical, and virtual to physical
- Evaluate expressions
- Control execution both in real and protected mode
- Set software breakpoints on execution addresses
- Set hardware breakpoints on execution and data addresses
- Disassemble memory
- Assembler with syntax guide

BENEFITS

Shortened Development Cycle

With P-MON386, you can use control constructs, which provide repetitive or conditional execution of P-MON386 commands, and you can make code patches by directly writing to memory. These features help to shorten the development cycle by easily isolating software bugs and by quickly testing program changes.

Improved Debugging Productivity

With the P-MON386 monitor you can display and modify program variables. In addition, it allows you to define, display, modify, and remove debug objects (such as break registers).

More Reliable Software

The control constructs can be used to repeatedly generate test values, execute the program with input values, and record the results. Running more comprehensive tests yields more reliable software.

Easy to Learn and Use

An extensive command language, which is similiar to block-structured languages such as PL/M, Pascal, and C, is very easy to use in an interactive debug session. The operators and control constructs are similiar to those in the C programming language.

The syntax guide is extremely helpful in constructing P-MON386 commands and considerably shortens the learning cycle. The ability to define literallys allows the user to extend and tailor the command language to suit individual needs.

FEATURES

Processor/Memory Examination and Modification

80386 registers can be accessed mnemonically (e.g., EAX) with the P-MON386 software. Data can be displayed or modified in one of four bases: hexadecimal, decimal, octal, or binary. Program code can be disassembled and displayed as 80386 assembly instruction mnemonics.

Execution Control

The GO command is used to begin execution or to resume execution at the current execution point or at a specified address. The P-MON386 monitor provides the ability to break on execution addresses using either software code patch breaks or hardware debug register breaks provided on the 80386 chip. The software breaks allow breaks on instruction execution only. These breakpoints can be placed on the RAM-based program code. The hardware register breaks allow recognition of the following conditions:

- An instruction boundary
- A data write to a user-specified linear or virtual address
- A data access at a user-specified linear or virtual address

Program Stepping

With the P-MON386 monitor, you can single step through program code by machine-level instructions, line numbers, or procedures.

Symbolic Debugging

The P-MON386 software takes advantage of the special debug information provided by Intel compilers, providing superior debug ability. It allows the software developer to examine or modify memory locations using symbolic references. A symbolic reference is a program procedure name, line number, or program label that corresponds to a location in the program space. Symbolic debugging allows you to work in the context of the original program, helping to meet the most critical schedules.

The P-MON386 monitor maintains a virtual table for program symbols making it possible for the table to exist without fitting entirely into the host RAM memory.

Debug Procedures

Debug procedures (PROCs) are named, user-defined groups of the P-MON386 commands. They can be stored on disk and recalled in later debugging sessions, thus saving you from having to re-enter commands.

One advantage of PROCs is that they allow you to automate the software testing process. A PROC may repeatedly generate test values, execute the user program with varied input values, and record the results. PROCs aid in the development of comprehensive batch tests.

Disassembler/Assembler

The P-MON386 monitor allows you to disassemble and display user target memory contents in the 80386 assembly instruction mnemonics.

The assembler allows 80386 assembler code to be interactively placed in program memory. The assembler provides syntax guidance to help remember seldom used mnemonics and operand syntax.

Human Interface

The P-MON386 software includes features to help enter commands, set up the debug environment, and display command options. Also, there are features similar to those found in the UNIX*/XENIX operating system environment. These features include input/output redirection, command piping and escaping from the shell. The P-MON386 software has a set of mathematical operators and control constructs similar to those found in Intel's I²ICE™ emulator and the C programming language.

Coprocessor Support

The P-MON386 commands provide access to the 80287 and 80387 coprocessor's stack, status registers, and flags. In addition, the disassembly and assembly commands extend to the math coprocessor's instructions and data types.

Language Support

The P-MON386 supports all 8086 absolute and 80286/80386 bootloadable Intel OMFs. These include the following Intel languages:

ASM86	ASM286	ASM386
PL/M-86	PL/M-286	PL/M-386
C-86	C-286	C-386
Pascal-86	Pascal-286	
FORTRAN-86	FORTRAN-286	

SPECIFICATIONS

Host Specifications

The following lists the minimum host requirements for the Intel 286/310 and the IBM PC AT.

- Intel System 286/310—2 MB of RAM, 40 MB fixed disk, at least one floppy disk drive, and the XENIX 286 operating system (version 3.0 with Update 4).
- IBM PC AT—2.0 MB of RAM (Lotus*, Intel, and Microsoft† extended memory specifications) (Intel's ABOVE Board with 1.5 MB is required), 20 MB fixed disk, at least one floppy disk drive, a serial interface, and the DOS operating system (version 3.1 or higher).

Required System Resources

P-MON386 monitor requires exclusive use of the 80386's on-chip debug registers, INT1, and the trap flag.

*UNIX is a trademark of Bell Labs.

Target System Requirement

The P-MON386 is designed to be used in any 80386-based target system that includes the following:

- At least 80K bytes of EPROM space
- At least 32K bytes of RAM space
- A serial communications interface

The starting locations for the software loaded into the EPROMs and RAM space is user configurable.

Other Required Tools/Systems

The P-MON386 comes pre-configured on diskettes for Intel SBC386/20 board which uses Intel 8251 and 8254 devices at 1.23 MHz clock rate. Baud rate support is from 1200 bauds to 19.2 Kbauds. P-MON386 also comes pre-configured to support the Intel's 386/100 board. P-MON386 is available for user-configurable applications as well.

Example drivers are included in User's Guide to aid in writing a custom driver. The following are required to write a custom driver, link it with the monitor code, build the target code, and burn EPROMs:

For writing a custom driver,

- ASM-286 or ASM-386 or PL/M-386 or C-386
- RLL-386
- Intel System 286/310 running XENIX Rel 3. Update 4 to link driver and build the target code or IBM PC AT running DOS 3.1 or higher to link driver and build the target code.

For burning EPROMs,

- Intel PROM Programmer
- iPPS version 2.1 or later version software (hosted on DOS or iNDX)

Shipping Media

Host software
5¼" 360KB TAR format and
5¼" 360KB DOS format diskettes

Target firmware
5¼" 360KB DOS format diskettes

ORDERING INFORMATION

Order Code	Description
PMON386	The P-MON386 monitor package includes host and target software diskettes and a user's guide. The host software is shipped on 5¼" 360KB TAR format and 5¼" DOS 360KB format diskettes and the target firmware is shipped on 5¼" 360KB DOS diskettes, Single user. (Requires Class I license)
PMON386020R10	Same as PMON386 except it is preconfigured to support Intel's SBC386/20 board. Target and host software diskettes and EPROMs are included.
PMON386100	Same as PMON386020 except configured for Intel's SBC386/100 board.
PMON386COPY	Right to make additional 8 copies.
PMON386SRC	P-MON386 source code.
PMON386	P-MON386. License to incorporate. Class II/Class III license.
PMON386RF	P-MON386. Royalty fees.
PMON386BY	P-MON386. Incorporation fee buyout.
PMON386HSW	Only host software.

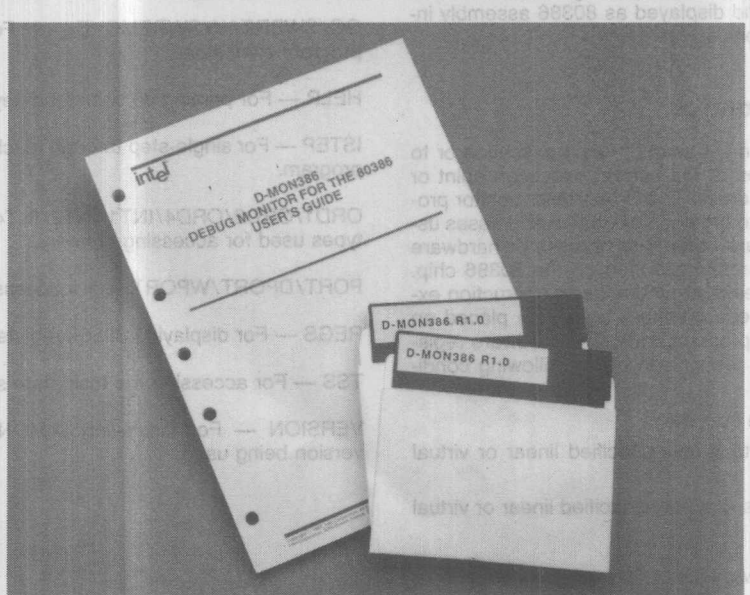
Purchase of any of these options requires the execution of a standard Intel Master Software license. The specific rights granted to users depends on the specific option and the license signed.

D-MON386
Debug Monitor 386

- **Unhosted Monitor Configurable on Any 80386-Based Board**
- **Provides Program Execution Control, Including Software Breakpoints and Single-Step Execution through Assembly Level Instructions in Both Real and Protected Mode**
- **Supports Four On-Chip Breakpoints to Recognize Instruction Execution Addresses or Data Access Addresses**
- **Allows User to Examine and Modify Memory, I/O Ports, and 80386 Registers**
- **Allows User to Examine Descriptor Tables, Task State Segment, and Page Table**
- **Allows Disassembly of Memory in 86/286/386 Instruction Mnemonics**
- **Supports Virtual, Linear, and Physical Addressing**
- **Supports Real, Protected, and Page Protected Modes of 80386**
- **Pre-Configured for Intel SBC386/20 and SBC386/100**
- **Provides Ten 32-Bit Scratch Registers for Storing Intermediate Values During the Debug Process**
- **Provides User-Friendly Human Interface with Command Line Editing and Command History**

The D-MON386 is an unhosted EPROM based software debug monitor for 80386-based systems and provides system level debug support. D-MON386 does not require a host system for its operation; it can access and control all of the 80386 visible user-hardware resources without any assistance from an operating system. With the help of this monitor, you can set hardware and software breakpoints, examine/modify memory and processor registers, and control program execution. This monitor can be configured to run on any 80386-based target board with a user-supplied communication driver and hardware initialization routine.

The D-MON386 package includes the diskettes that contain the target software to be burned into EPROM for 80386-based target board, and the D-MON386 User's Guide.



166187-1

October 1987
Order Number: 166187-003

FUNCTIONAL DESCRIPTION

Overview

D-MON386 provides design aid and debug support for Intel customers who design software to run on the 80386. D-MON386 allows you to perform the following tasks:

- Examine and modify memory, I/O ports, 80386 registers, and display descriptor tables, task state segment, and page table
- Control execution both in real and protected mode
- Set four on-chip breakpoints to recognize instruction execution addresses or data access addresses
- Set software program breakpoints
- Single-step through 80386 instruction execution
- Disassemble program code in 86/286/386 instruction mnemonics
- Evaluate expressions

Processor/Memory Examination and Modification

80386 registers can be accessed mnemonically (e.g. EAX) with the D-MON386 software. Data can be displayed or modified in one of four bases: hexadecimal, decimal, octal, or binary. Program code can be disassembled and displayed as 80386 assembly instruction mnemonics.

Execution Control

The GO command is used to begin execution or to resume execution at the current execution point or at a specified address. The D-MON386 monitor provides the ability to break on execution addresses using either software code patch breaks or hardware debug register breaks provided on the 80386 chip. The software breaks allow breaks on instruction execution only. These breakpoints can be placed on the RAM-based program code. The hardware register breaks allow recognition of the following conditions:

- An instruction boundary
- A data write to a user-specified linear or virtual address
- A data access at a user-specified linear or virtual address

D-MON386 allows you to single-step through your program code by referencing machine-level instructions.

Disassembler

The D-MON386 monitor allows you to disassemble and display user target memory contents in the 80386 assembly instruction mnemonics.

Formatted Displays

The D-MON386 allows you to display all of the 80386 visible hardware resources and pre-defined data structures in easy-to-read formats. These displays include contents of the global descriptor table (GDTs), the local descriptor table (LDTs), the interrupt descriptor tables (IDTs), the task state segments (TSSs), the extended flags register, the segment registers, the 80386 registers, Control Registers (CREGs), page directory, and page tables.

Debugger Command Language

ASM — For disassembling code.

BASE — For changing the default base for command line entry.

COUNT — For repetition of commands or blocks.

DT/GDT/LDT/IDT — For allowing display of descriptor tables.

EVAL — For evaluating expressions.

GO/SWBREAK/SWREMOVE — For controlling program execution.

HELP — For printing all command keywords.

ISTEP — For single-step through machine level user program.

ORD1/ORD2/ORD4/INT1/INT2/INT4 — Data types used for accessing memory.

PORT/DPORT/WPORT — For accessing I/O ports.

REGS — For displaying 80386 registers.

TSS — For accessing the task state segment.

VERSION — For displaying D-MON386 software version being used.

SPECIFICATIONS

Required System Resources

D-MON386 monitor requires exclusive use of the 80386's on-chip debug registers, INT1, and the trap flag.

Target System Requirement

The D-MON386 is designed to be configured on any 80386-based target system that provides the following:

- At least 128K bytes of EPROM space
- At least 64K bytes of RAM space
- A serial communications interface

The starting location for the software loaded into the EPROMs and RAM space is user configurable.

Other Required Tools/Systems

The D-MON386 comes pre-configured for Intel SBC386/20 board which uses Intel 8251 and 8254 devices at 1.23 MHz clock rate. Baud rate supported is 9600. D-MON386 also comes pre-configured for Intel's SBC 386/100 board. D-MON386 is available for user-configurable applications as well.

Example drivers are included in User's Guide to aid in writing a custom driver. The following are required to write a custom driver, link it with the monitor code, build the target code, and burn EPROMs:

For writing a custom driver,

- ASM-286, ASM-386 or PL/M-386 or C-386
- RLL-386
- Intel System 286/310 running XENIX Rel 3 Update 3 to link driver and build the target code

  Alternately PC AT/DOS hosted software tools can be used to write a custom driver

For burning EPROMs,

- Intel PROM Programmer
- iPPS version 2.1 software or higher (hosted on DOS or iNDX)

ORDERING INFORMATION

Order Code	Description
DMON386R1.0	The DMON386R1.0 monitor package includes the target software diskette containing linkable object code and a user's guide. Single user. (Requires Class I license)
DMON386020R1.0	Same as DMON386R1.0 except pre-configured for Intel's SBC 396/020 board.
DMON386100R1.0	Same as DMON386020R1.0 except pre-configured for Intel's SBC 386/100 board.
DMON386COPY	Right to make additional 8 copies.
DMON386SRC	DMON386R1.0 source code.
DMON386RO	DMON386R1.0. License to incorporate. Class II/Class III license.
DMON386RF	D-MON386R1.0. Royalty fees.
DMON386BY	D-MON386R1.0. Incorporation fee buyout.

Purchase of the D-MON386 package includes a 90-day support service, which also includes the Software Problem Report Service.

Purchase of any of these options requires the execution of a standard Intel Master Software license. The specific rights granted to users depends on the specific option and the license signed.

Shipping Media

PC-DOS Format DS/DD 5¼" diskette

iPAT™-86/88
Performance Analysis Tool

- **Provides Symbolic Real-Time Software Analysis for Software Running on MAX MODE 8086 and 8088 Systems**
- **Supports Absolute and Load-Time Locatable Programs**
- **Collects 100% of Software Execution Transparently, Supporting the 8086 to 10 MHz and the 8088 to 8 MHz**
- **Displays Histograms and Performance Tables for Up to 125 Modules, Procedures, Lines, or User-Defined Ranges**
- **Shows Code Execution Coverage For All Modules, Procedures, Lines, or Ranges in Up to 252K Anywhere in the 8086 or 8088 Address Space**
- **Supports Symbolics for All Block-Structured 86 High-Level Languages — Such as C, PL/M, Pascal, Ada, FORTRAN and Assemblers**
- **Provides Distributions of Latency to Hardware Interrupts**
- **Presents Simple User Interface With Function Keys and Color or Monochrome Graphic Displays**
- **Hosted Serially on PC-DOS 3.xx Systems**
- **Supports Standard 40-Pin DIP Packaging (a 40-Pin DIP to 44-Pin PLCC Convertor is Available Separately)**

iPAT™-86/88, Intel's Performance Analysis Tool for 8086 and 8088 microprocessor applications, provides a new technology for software analysis in real-time systems. iPAT opens up the mystery of software-driven systems, revealing the real-time execution activity of software in the prototype environment non-intrusively and with 100% collection. With this understanding the developer can control the performance and reliability of real-time applications more efficiently and effectively than has been possible in the past. iPAT-86/88 supports component-, board-, and system-level products based on the 8086 and 8088.

Mode: PROFILE
PTIMEBASE = 200 ns
Include Calls
Status: OK

ABS = FALSE
HISTO = TIME
SORT = ADDRESS
FILTER = FALSE

Event	Time	0% 5 10 15 20 25
:CDU.GET__LOADING	471 ms	
FIND__3D__POSITION	616 ms	
READ__SENSORS__	589 ms	
GET__AIRSPEED__	0 ns	
GET__THROTTLE__	380 ms	
GET__AILERONS__	120 ms	
READ__SENSORS	60 ms	
GET__FLAP__SETTINGS	132 ms	
CALC__COURSE	312 ms	
REFRESH__DISPLAY__	740 ms	
GET__PILOT__RESPONSE	189 ms	
SET__THROTTLE__	80 ms	
SET__AILERONS__	309 ms	
SET__RUDDER__	1800 ns	
SET__FLAPS__	176 ms	
Total	4174 ms	0% 5 10 15 20 25

August 1987
Order Number: 280709-001

INTRODUCTION TO SOFTWARE ANALYSIS

The size and complexity of software have increased with each new generation of microprocessors. As a result it has become increasingly important - and difficult - to optimize software and to ensure its reliability. For embedded as well as reprogrammable systems, real-time software analysis with Intel's iPAT Performance Analysis Tool answers these needs.

iPAT software analysis can be used for all types of MAX MODE 8086 and 8088 applications (MAX MODE is a hardware design mode of the 8086 and 8088. It is used in all 8086 and 8088 systems, nearly all board-level products, such as MULTIBUS I boards, and most component-level designs). For example, iPAT-86/88 can be used to profile the performance characteristics of a load-time locatable application running on an operating system, or of an absolutely-located embedded application running on bare hardware in an 8086 or 8088 embedded application. The developer uses iPAT to obtain histograms of overall performance, subroutine characteristics, and interactions with interrupt routines, external hardware, or with the operating system. These applications then can be speed-tuned based on real-time execution information. At the acceptance phase of development of these products, evaluators can use iPAT to monitor code execution by test cases and to prove ability to meet response time specifications.

An iPAT-86/88 software analysis system consists of an iPAT core module, an iPAT-86/88 probe, and iPAT host software for PC AT or PC XT hosts running PC-DOS 3.xx. The iPAT probe uses 8086 and 8088 chips that operate transparently at clock speeds to 10 MHz and 8 MHz, respectively, in the user's application. The iPAT core module is the same as used with other iPAT probes and interfaces, such as iPAT-286 and iPAT/I2ICE™.

The iPAT-86/88 user interface provides function keys and context-sensitive prompt lines for command input, and graphic displays for output. Software analysis with iPAT is easily learned and understood. Session activity, including all histograms and tabular displays, can be logged to disk files and printed for documentation.

Optimizing Software

Optimizing real-time software performance means maximizing execution speed on the CPU and speed-tuning for real-time I/O interactions without sacrificing functionality or reliability. To increase speed, implementation characteristics must be compared to design expectations to isolate bottlenecks.

In the past, this comparison has been a hit and miss process. The engineer confronted the microprocessor as a black box. Real-time software execution characteristics inside the chip remained a mystery.

Without iPAT software analysis, the software engineer might conduct a walkthrough of the various paths and I/O in the source code, and then make educated guesses as to where bottlenecks are likely to occur. However, the difficulty of this approach increased rapidly as the complexity of the application increased. Alternatively, the software engineer might stub parts of the software for execution on the computer system used for development, insert count and print statements in source code, and then build and analyze a special version of the software. In addition to being time-consuming, this approach was useless for embedded designs. An embedded design's real-time I/O characteristics cannot be meaningfully simulated on the development computer system.

With iPAT software analysis, the engineer can quickly and easily show real execution time for program modules, procedures, lines, or user-defined symbolic ranges in the actual 8086 and 8088 embedded or reprogrammable system environment. Software analysis with iPAT-86/88 provides histograms and tabular displays with the high-level names for program partitions, such as modules and procedures. iPAT automatically builds an internal data structure for correspondence of symbols and addresses using 8086 object module format information (OMF86). This information is incorporated in executable files built with Intel software tools. (For object code built with software tools from other vendors, conversion is possible using utilities supplied with the iPAT host software package.)

Software analysis with iPAT is non-intrusive and carried out in real-time. iPAT does not sample program execution to build a statistical model; rather, it collects and compares all addresses in real-time.

Software teams coding in assembly language to increase program speed will now find they can change their methods to be more productive. They can write applications in high-level languages for easier expression of complex algorithms and easier code maintenance. Many parts of an application may have no significant impact on overall execution speed. Those parts that do can be isolated through iPAT software analysis. The algorithms can then be improved, or the code converted into special-purpose assembly language routines to improve execution speed.

Optimizing Hardware Interactions

iPAT-86/88 also enables the developer to examine the timing characteristics of interactions between

hardware devices and software. By this means the developer can measure real-time characteristics that are critical to embedded applications, such as the latency between detecting a sensor condition for an aircraft and executing software routines that respond to the condition.

Ensuring Reliability

To ensure application reliability, developers typically devise unit-level and system-level test suites. A theoretical understanding of the design is used to generate test suite data. Then the application is executed and its output is examined. Depending on application complexity and testing time, a higher or lower statistical confidence level for the thoroughness of testing is obtained.

'Black box' testing of this sort can only evaluate inputs and outputs; not the actual execution of application code in real-time, which remains a matter of inference. Thus, in the past, evaluators could not be confident that their test cases actually executed all the code. Errors in untested code could not be detected because it was not known that the code was not executed by the test cases. Many developers were forced to trade off lengthening test times against the need to hit market windows, or to limit innovation.

With iPAT real-time code coverage analysis, the engineer can quickly and easily identify which lines, procedures, and modules are executed by test cases and which are not. Armed with this information, the evaluator determines incremental test input to hit the unexecuted portions of the application. The result is more efficient and effective testing, and greater freedom to innovate without sacrificing quality.

Impact of Real-Time Software Analysis on Development

Developers must introduce microprocessor-based products into competitive environments and product acceptance processes where performance and reliability are critical to success. iPAT provides a means of controlling performance and reliability to meet design expectations and a higher level of productivity in several phases of the product cycle.

In addition to supporting initial product development, iPAT can be used during the maintenance cycle to increase the quality of enhancements and reduce their cost. For example, using iPAT code coverage analysis to execute existing test suites for evaluation of subsequent releases enables engineers to determine more quickly and easily the incremental test cases needed to produce full code coverage. Developers also can more easily isolate and correct the performance bottlenecks that frequently result from enhancements.

The front end of the development cycle also can be improved by iPAT's real-time software analysis. New product designs often re-use existing code modules to reduce development costs. If good performance and reliability parameters have been built in and are known from previous software analysis, then the work of the designer who re-uses modules will be more productive.

Finally, real-time software analysis encourages engineers to engage in 'what-if' thinking. They can ask, 'What if this portion of the application were designed this way, or used this algorithm?' Alternatives can be compared to determine which is most efficient in the actual application environment.

iPAT™-86/88 OPERATION

Gaining real-time software analysis information in symbolic terms is simple with iPAT-86/88. The example here assumes an absolutely-located 8086 embedded application running on a real-time operating system kernel, but applies similarly to applications loaded by an operating system.

Obtaining Execution Histograms

To get started, the user replaces the 8086 chip in the embedded application with the iPAT-86/88 probe using an 8086 chip, switches on power for the iPAT system, and starts the 8086 target system. The next step is to load program symbolics into iPAT on the host PC system, instruct iPAT to profile, run the application in the target system, and display the results on iPAT. iPAT commands issued would be:

```
LOAD <my_prog.86>
PROFILE PROC INC :cdu

   /* profile procedures in the CDU
      module with inclusion of time
      in other subroutine calls */

   /* run application in target
      system */

DISPLAY
```

The default display is a histogram (see Fig. 1), ordered from low to high addresses. Histogram bars indicate relative time consumption. The cumulative execution times for each procedure also are shown.

Example: The procedure REFRESH__DISPLAY__ has consumed 740 ms or about 17% of total execution time.

Mode: PROFILE
PTIMEBASE = 200 ns
Include Calls
Status: OK

ABS = FALSE
HISTO = TIME
SORT = ADDRESS
FILTER = FALSE

Event	Time	0% 5 10 15 20 25
:CDU.GET__LOADING	471 ms	
FIND__3D__POSITION	616 ms	
READ__SENSORS__	589 ms	
GET__AIRSPEED__	0 ns	
GET__THROTTLE__	380 ms	
GET__AILERONS__	120 ms	
READ__SENSORS	60 ms	
GET__FLAP__SETTINGS	132 ms	
CALC__COURSE	312 ms	
REFRESH__DISPLAY__	740 ms	
GET__PILOT__RESPONSE	189 ms	
SET__THROTTLE__	80 ms	
SET__AILERONS__	309 ms	
SET__RUDDER__	1800 ns	
SET__FLAPS__	176 ms	
Total	4174 ms	0% 5 10 15 20 25

Figure 1. Histogram with Include Calls

Information provided at the top left of the screen indicates profiling at a granularity of ± 200 nanoseconds with inclusion of any time spent in called subroutines. Information provided at the top right indicates further display options (default values are shown). For example, the information is sorted in order of ascending addresses, there is no filtering of what is shown in the display, and 'background' time is not included (ABS = FALSE).

Displays have default settings but are flexible. For example, the user can change the display to show descending order by relative time consumption and to show only those procedures consuming more than 600 milliseconds. Instead of displaying results once, typing DISPLAY UPDATE 0 would produce a continuously updated display:

```
SORT = TIME
FILTER = TIME > 600 MS
DISPLAY UPDATE 0
```

The display now continuously refreshes with information only for those procedures that have accumulated more than 600 milliseconds of execution time.

Greater detail is obtained (see Fig. 2) by deselecting the histogram form of display:

```
HISTO = FALSE
DISPLAY
```

Example: GET__THROTTLE__ was executed 49 times. Total execution time was 380 ms, with 7755 μs as the mean, 2002 μs as the minimum, and 16 ms as the maximum timing over the 49 executions.

Mode: PROFILE
PTIMEBASE = 200 ns
Include Calls
Status: OK

ABS = FALSE
HISTO = FALSE
SORT = ADDRESS
FILTER = FALSE

Event	Count	Time	Time Min	Time Avg	Time Max
:CDU.GET__LOADING	3	471 ms	157 ms	157 ms	157 ms
FIND__3D__POSITION	14	616 ms	14 ms	44 ms	181 ms
READ__SENSORS__	31	589 ms	7000 μs	19 ms	21 ms
GET__AIRSPEED__	0	0 ns	0 ns	0 ns	0 ns
GET__THROTTLE__	49	380 ms	2002 μs	7755 μs	16 ms
GET__AILERONS__	26	120 ms	1110 μs	4615 μs	11 ms
READ__SENSORS	14	60 ms	1000 μs	4285 μs	9020 μs
GET__FLAP__SETTINGS	12	132 ms	9000 μs	11 ms	34 ms
CALC__COURSE	26	312 ms	7000 μs	12 ms	14 ms
REFRESH__DISPLAY__	2	740 ms	72 ms	370 ms	668 μs
GET__PILOT__RESPONSE	3	189 ms	63 ms	63 ms	63 ms
SET__THROTTLE__	2	80 ms	40 ms	40 ms	40 ms
SET__AILERONS__	3	309 ms	103 ms	103 ms	103 ms
SET__RUDDER__	1	1800 ns	1800 ns	1800 ns	1800 ns
SET__FLAPS__	11	176 ms	14 ms	16 ms	17 ms
Totals	197	4174 ms			

Figure 2. Profile Include Calls Tabular Display

Examining Function Timing

Most applications have specifications for the timing of certain functions, such as worst-case timing to update displays, response to hardware interrupts, or transition timing characteristics. These types of timing can be characterized with specialized iPAT analysis modes.

In the following example, the DURATION mode is used to obtain a distribution of the time required to refresh a cockpit display screen. This function is initiated in the CDU module at line #145 and completed with a call to the BIOS module's WRITE__TTY function. For convenience, the engineer can define a range name for this part of the program's functional flow. The engineer can then set up iPAT to characterize the duration of this range. In the example (see Fig. 3), logarithmic scaling of timing bins from 200 nanoseconds to 1 second is specified:

```
DEFINE RANGE tty_update_time =
                    :CDU#145 TO :BIOS.WRITE_TTY

DURATION tty_update_time LOGB 200 NS
                                    TO 1SEC

   /* run application */
```

The distribution peak shows that, generally, execution was very fast, but a second peak and outliers are evident. These may be exceeding performance requirements.

Mode: DURATION
Event: TTY__UPDATE__TIME
Time Range: 200 ns to 1 sec
PTIMEBASE = 200 ns
Status: OK

SELECT = INNER
HISTO = COUNT

Type: LOGB

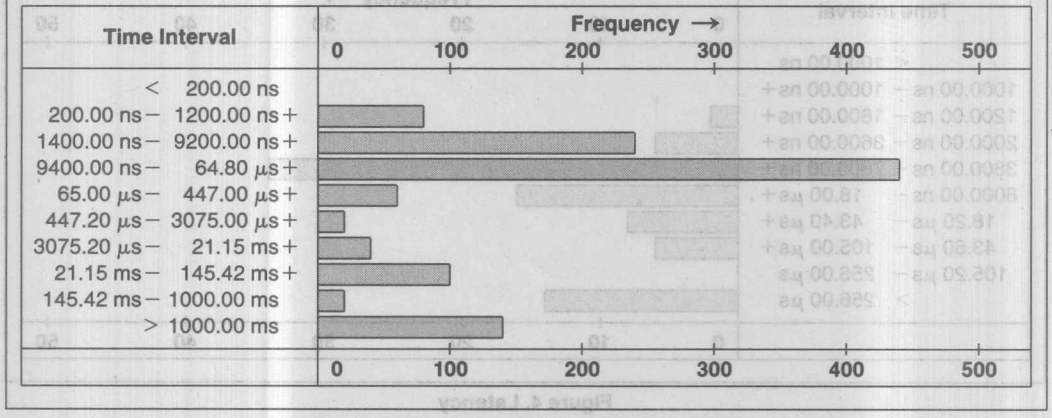


Figure 3. Duration of TTY__UPDATE__TIME

Other performance requirements for this application might define worst-case latencies to various hardware interrupts. For example, a worst-case latency of no more than 256 microseconds from a hardware interrupt issued at ground contact by the landing gear until code is executed to shut down the aircraft's jet engines. Using the LATENCY analysis mode of iPAT, the interrupt latency can be characterized. In the example (see Fig. 4), logarithmic scaling of timing bins from 1000 nanoseconds to 256 microseconds is specified:

```
LATENCY interrupt TRANS
   :eng_mgr.stop_thrust_
   LOGB 1000 NS TO 256 US

   /* attach iPAT-86/88 interrupt
      clip to device issuing
      interrupt and run application
      using developer's test
      harness */
```

Typical latency is slightly less than 1 μs but in about 13 cases exceeded the worst-case requirement of 256 μs.

Mode: LATENCY
Event: INTERRUPT TRANS :ENG_MGR.STOP_THRUST_
Time Range: 1 μs to 256 μs
PTIMEBASE = 200 ns
Status: OK

SELECT = INNER
HISTO = COUNT
Type: LOGB

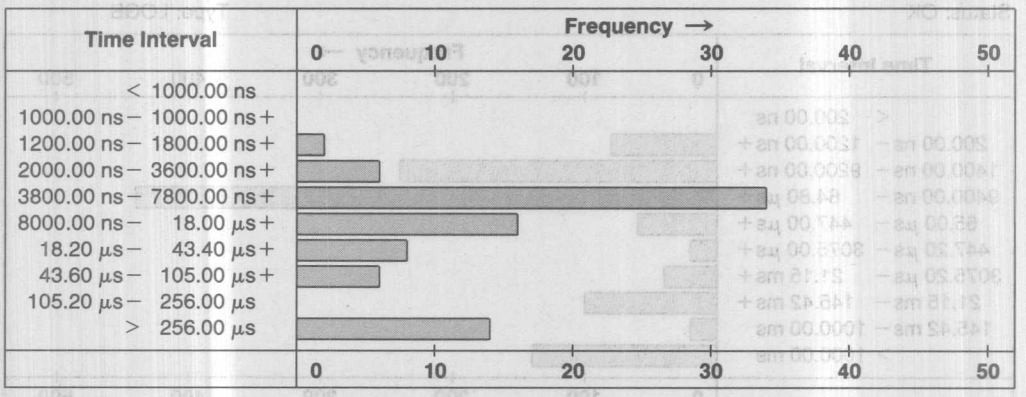


Figure 4. Latency

In many applications timing requirements also exist for the linkage between certain functional parts of the program. The LINKAGE mode of iPAT measures procedure interaction of this type. In the example (see Fig. 5), linkage between a procedure that calculates flight course corrections and several other program functions is characterized:

```
LINKAGE :cdu.calc_course LINK :io_sim.update, &
:cdu.calc_course LINK :libs.trig_fxns, &
:cdu.calc_course LINK :libs.vector, &
:cdu.calc_course LINK :chart.lax_calc
```

Timing data suggests that the interaction of the CALC__COURSE routine with the TRIG__ FXNS is a likely source of bottlenecks.

Mode: LINKAGE
PTIMEBASE = 200 ns
Status: OK

Event	Count	Time	Time Min	Time Avg	Time Max
.:CDU.CALC__COURSE-:IO__SIM.UPDATE	7	92 μs	13 μs	13 μs	13 μs
.CALC__COURSE-:LIBS.TRIG__FXNS	7	3300 ms	120 ms	471 ms	908 ms
.CALC__COURSE-:LIBS.VECTOR	6	1264 ms	200 ms	210 ms	640 ms
.CALC__COURSE-:CHART.LAX__CALC	7	300 μs	38 μs	43 μs	45 μs

Figure 5. Linkage Display

Identifying Code Coverage

At the test and acceptance phase, engineers need to achieve as high a level of confidence as possible in as little testing time as possible. Code coverage analysis with iPAT is a means to accomplish this, without having to rely entirely on statistical models. COVERAGE mode identifies parts of the code that have been executed by tests and those that have not. Incremental testing inputs then can be determind, or sections of 'dead code' removed from the program. Code coverage data also provides a means for the evaluator to document testing results.

In the example (see Fig. 6), code coverage results are shown at procedure level. The commands to obtain these results were:

```
COVERAGE :cdu
   /* now execute test cases on the
      application system. /*
DISPLAY
```

Procedures that were executed by the test case inputs are indicated with a mark next to them. The engineer can log code coverage results to disk files. The code coverage data structure internal to iPAT is not cleared with each new set of test inputs, unless CLEAR is requested. Thus incremental results are added for additional tests.

The upper right part of the display indicates that procedure-level results are shown. This is the default. Without re-executing tests, the display can be changed to line-level or module-level results. All modules, procedures, and lines can be displayed. The only limitation is that all areas specified for code coverage must comprise a total of no more than 252K of program memory, which need not be contiguous. Memory space(s) to be covered are automatically programmed by iPAT from the user's symbolic specifications, such as COVERAGE :mod__a to :mod__n,:mod__p, :mod__u.

Utilities for producing address-by-address boolean maps of code coverage or instruction-level output are also provided with iPAT.

The procedure READ__SENSORS__ is marked as having been executed; GET__AIRSPEED__ is not. 80% of the procedures were executed. Note that this data can be redisplayed at the line level.

Mode: COVERAGE SHOW = ALL PROC

Exec	Event	Exec	Event	Exec	Event
:CDU					
■	GET__LOADING	■	GET__AILERONS__	■	GET__PILOT__RESPONSE
■	FIND__3D__POSITION	■	READ__SENSORS		SET__THROTTLE__
■	READ__SENSORS__	■	GET__FLAP__SETTINGS	■	SET__AILERONS__
	GET__AIRSPEED__	■	CALC__COURSE		SET__RUDDER__
■	GET__THROTTLE__	■	REFRESH__DISPLAY__	■	SET__FLAPS__

EXEC: 12 (80.0%) NOEXEC: 3 (20.0%) Total: 15

Figure 6. Coverage Display

USAGE WITH AN IN-CIRCUIT EMULATOR

iPAT-86/88 monitors real-time software activity symbolically and passively to provide the software engineer with an instrument analogous to an oscilloscope for the hardware engineer. But in some development environments, interaction with the prototype is needed: for example, downloading code or writing data to an incomplete prototype. iPAT-86/88 has the flexibility required for use in conjunction with Intel's I2ICE-8086/88 emulator system. Under this scenario, the emulator and iPAT have separate PC hosts, but share the in-circuit emulator's 8086/88 probe. By this means, unit-level tests can be generated procedurally by the in-circuit emulator and analyzed by the iPAT system.

HOST SYSTEM REQUIREMENTS

An IBM PC AT or PC XT system running PC-DOS 3.xx with at least 512K bytes of available memory in RAM.

An RS232 serial port that operates at 300, 1200, 9600 (default), or 19200 baud must be available on the host system.

ANALYSIS MODES

Modes with Multiple Partitions

The following iPAT modes collect execution information for multiple modules, procedures, lines, user-defined ranges, or absolute address partitions.

Profile mode: Collects time and count profiles for up to 125 program partitions (modules, procedures, line numbers, or user-defined ranges). Time information includes absolute cumulative, minimum, maximum, and average execution time for each partition. Profile information can be collected using any of three options: INC, NOINC, or EXC. These options set profile mode to include calls, not include calls, or show range activity under execution that is broken up by interrupts and called routines. The user can display information in histogram or tabular form, and sort by address, cumulative time, or cumulative count. Filters can be set to select a subset of partitions that have specified characteristics.

Linkage mode: Collects time and count information for the linkage interaction of up to 63 events, such as procedure to procedure activity, including recursive activity. The user can display information in histogram or tabular form, and sort by address, cumulative time, or cumulative count.

Usage mode: Collects time and count information about usage of one 'used' event by up to 63 other 'user' events, such as a subroutine that is called by many subroutines. The user can display information in either histogram or tabular form, and sort by address, cumulative time, or cumulative count.

Coverage mode: Provides coverage information for up to 63 4K-byte blocks of address space, which need not be contiguous and are mappable anywhere in the 8086 and 8088 physical address space. Coverage areas are set automatically by specifying partitions for coverage with module, procedure, line number, or absolute address boundaries. Boolean coverage maps are automatically converted into partition names such as modules, procedures, line numbers, or user-defined ranges by the iPAT host software for user display. Coverage information is cumulative, incremental, and can be recorded to a user-specified file, or recalled, by using iPAT's SAVE and RECALL commands.

Modes With Multiple Timing Bins

The following iPAT modes organize data into timing bins for a single partition. The partition can be a procedure or starting-ending pair of events bounding the partition). Users can select bin granularity (4, 8, 16, or 32 bins), linear or logarithmic scaling, as well as the lower and upper bounds for the range of bins. The distribution of execution times can be displayed as a histogram or as a table, including minimum, maximum, and cumulative execution times.

Duration mode: Collects a distribution of execution times across a range of timing bins for a single partition.

Latency mode: Collects a distribution of latency from a hardware interrupt to a software execution event, or between interrupts, across a range of timing bins. A clip line from the iPAT probe to the source of the interrupt signal is used to inform iPAT of asynchronous events external to the software execution address stream.

Windowed-event count mode: Collects count data concerning an event that occurs within a specified event window.

FUNCTIONALITY

Partition specification: Program partitions can be specified using program symbolic names, user-defined symbolic names, or absolute addresses. Symbols anywhere in the 8086 and 8088 address space can be used. Thus code loaded dynamically, such as overlays, must not result in multiple symbolic meanings for an identical physical address space in the 8086 or 8088.

Symbolic Information: Program symbolics for modules, procedures, and line numbers are loaded by iPAT host software using OMF86 (8086 object module formats) in absolute or load-time locatable files built with Intel 86 compilers using the DEBUG option and Intel linkers, locaters, builders, and binders. Utilities are included for converting non-Intel symbolic information to iPAT-loadable symbolics.

Speed: The iPAT-86/88 probe supports the 8086 to 10 MHz and the 8088 to 8 MHz, collecting instruction addresses at full processor speeds (however, sustained execution in a short loop may cause iPAT processing overflow).

Timebase: The iPAT data collection timebase is selectable at 200 nanoseconds, 1 microsecond, 10 microseconds, or 100 microseconds.

Display updates: An update frequency can be specified by the user at an interval in seconds.

Status: If time-count, bin-count, or FIFO stack overflow occurs, iPAT displays indicate the error status.

Session logs: The user can log part or all of an iPAT session, including displays and input/output in the iPAT environment, by using the LIST command to initiate recording to a specified log file and the NOLIST command to end recording.

Command input: All commands can be issued from iPAT's context-sensitive command line, which prompts the user with valid options. Commonly used commands also can be issued through predefined function keys. The user may define additional function keys and automatically include these in future sessions.

iPAT definitions: iPAT provides a powerful set of procedural constructs to bring greater flexibility and productivity to the engineer's software analysis. For example, partition specifications and analysis controls can be collected together in a 'SETUP' object and then edited, saved, and recalled. Display controls similarly can be defined as a TEMPLATE. LITERALLY and RANGE definitions can be created, saved, and re-used for a symbolic short-hand to abbreviate frequently repeated commands and to add high-level range names during analysis.

UTILITIES

Various utilities are supplied with the iPAT host software. These provide, for example, conversion of input files to a form offering symbolic functionality when loaded into iPAT, and conversion of output files to a form in which instruction-level coverage information can be generated by the user.

Microsoft* symbolics: MS*-LINK map file address information for modules, procedures, and lines may be converted to an OMF86-compatible, iPAT-loadable symbolics file.

Other non-OMF symbolics: Address/symbolic information for module, procedure, and lines generated by the user from other vendors' tools in a standard text file format also can be converted to OMF86-compatible, iPAT-loadable symbolics. After loading the converted information into iPAT, the user must be able to supply the bases to which code segments are loaded.

Boolean coverage map: A boolean map indicating all addresses covered and not covered during execution can be produced by saving iPAT code COVERAGE results to a file and then processing it through a utility.

Instruction-level coverage: An instruction-level coverage text file can be produced by processing a boolean map (see above) of address execution and a user-supplied file containing disassembly of the code memory image through a utility supplied with the iPAT-86/88 software. The disassembly file is a standard formatted text file produced, for example, by use of a software debugger or an in-circuit emulator from the memory image of the executable code.

SPECIFICATIONS

Physical Characteristics

iPAT-86/88 PROBE

Length: 15.3 cm (6 in)
Width: 10.2 cm (4 in)
Height: 2.86 cm (1.125 in)

iPAT CORE MODULE

Length: 35 cm (13¾ in)
Width: 21 cm (8¼ in)
Height: 4.5 cm (1¾ in)

iPAT POWER SUPPLY

Length: 28 cm (11 in)
Width: 20 cm (7¾ in)
Height: 11 cm (4¼ in)

AC power cord for the power supply: 3.0 m (10 ft)

Power-supply-to-core DC power cable: 1.8 m (6 ft), 10 conductor

iPAT probe-to-target cable: 10.2 cm (4 in)

iPAT probe-to-core cable: 0.9 m (36 in), 60 conductor

RS232 serial cables for connecting iPAT core to PC-XTor PC-AT host system: 3.7 m (12 ft)

Electrical Characteristics

— Worst-case incremental power requirements of probe in user's system: 720 ma.
— Selectable AC power source: 100V, 120V, 220V, 240V
— 47-63 Hz
— 2 amps (AC) at 100V or 120V; 1 amp at 220V or 240V

Environmental Requirements

— Operating temperature: 10 to 40 degrees C (50 to 104 degrees F)
— Operating humidity: Maximum of 85% relative humidity, non-condensing

ORDERING INFORMATION

Order Code	Description
iPAT88DOS	8086/88 probe, PC-DOS 3.xx host software, utility software, interrupt clip line, iPAT probe to iPAT Core interface cable, and PC XT/AT RS232 serial cables.
iPATCORE	iPAT core system, DC and AC power cables, and selectable 100V, 120V, 220V, 240V power supply.

*MS™-LINK and Microsoft™ are trademarks of Microsoft Corporation.

iPAT™-286
PERFORMANCE ANALYSIS TOOL

- **Provides Symbolic Real-time Software Analysis for Software Running on the 80286**
- **Supports Absolute and Load-time Locatable 8086 and 80286 Real Mode Programs, as Well as Boot-loadable and Single-task Loadable 80286 Protected Mode Programs**
- **Collects 100% of Software Execution Transparently and Supports the 80286 to 12.5 MHz**
- **Displays Histograms and Performance Tables for up to 125 Modules, Procedures, Lines, or User-defined Ranges**
- **Shows Code Execution Coverage for all Modules, Procedures, Lines, or Ranges in up to 252K Anywhere in the 80286 Address Space**
- **Supports Symbolics for 86 and 286 Assemblers and High-level Languages — Such as C, PL/M, Pascal, Ada, FORTRAN and Assemblers**
- **Provides Distributions of Latency to Hardware Interrupts**
- **Presents Simple User Interface with Function Keys and Color or Monochrome Graphic Displays**
- **Hosted Serially on PC DOS 3.xx Systems**
- **Support for PGA, LCC, and PLCC 80286 Packaging is Available**

iPAT™-286, Intel's Performance Analysis Tool for 80286 microprocessor applications, provides a new technology for software analysis in real-time systems. iPAT opens up the mystery of software-driven systems, revealing the real-time execution activity of software in the prototype environment non-intrusively and with 100% collection. With this understanding the developer can control the performance and reliability of real-time applications more efficiently and effectively than has been possible in the past. iPAT-286 supports component-, board-, and system-level products based on the 80286 in either real (86 Emulation) or protected (286 Native) mode operation.



280708-1

October 1987
Order Number: 280708-001

INTRODUCTION TO SOFTWARE ANALYSIS

The size and complexity of software have increased with each new generation of microprocessors. As a result it has become increasingly important - and difficult - to optimize software and to ensure its reliability. For embedded as well as reprogrammable systems, real-time software analysis with Intel's iPAT Performance Analysis Tool answers these needs.

iPAT software analysis can be used for a wide range of 80286 application types. As an example of two extremes, iPAT-286 can be used to profile the performance characteristics of a single-task loadable application running on an 80286 protected-mode operating system, or of an absolutely-located embedded application running on bare hardware in 80286 real mode in a dedicated board environment. The developer uses iPAT to obtain histograms of overall performance, subroutine characteristics, and interactions with interrupt routines, external hardware, or with the operating system. These applications then can be speed-tuned based on real-time execution information. At the acceptance phase of development of these products, evaluators can use iPAT to monitor code execution by test cases and to prove ability to meet response time specifications.

An iPAT-286 software analysis system consists of an iPAT core module, an iPAT-286 probe, and iPAT host software for PC AT or PC XT hosts running PC-DOS 3.xx. The iPAT-286 probe contains an 80286 bondout chip that enables the probe to operate transparently as an 80286 at any clock speed up to and including 12.5 MHz in the user's application. The bondout is a special version of the 80286 that supplies information required for iPAT analysis. The iPAT core module is the same as used with other iPAT probes and interfaces, such as iPAT-86/88 and iPAT/I2ICE™.

The iPAT-286 user interface provides function keys and context-sensitive prompt lines for command input, and graphic displays for output. Software analysis with iPAT is easily learned and understood. Session activity, including all histograms and tabular displays, can be logged to disk files and printed for documentation.

Optimizing Software

Optimizing real-time software performance means maximizing execution speed on the CPU and speed-tuning for real-time I/O interactions without sacrificing functionality or reliability. To increase speed, implementation characteristics must be compared to design expectations to isolate bottlenecks.

In the past, this comparison has been a hit and miss process. The engineer confronted the microprocessor as a black box. Real-time software execution characteristics inside the chip remained a mystery.

Without iPAT software analysis, the software engineer might conduct a walkthrough of the various paths and I/O in the source code, and then make educated guesses as to where bottlenecks are likely to occur. However, the difficulty of this approach increased rapidly as the complexity of the application increased. Alternatively, the software engineer might stub parts of the software for execution on the computer system used for development, insert count and print statements in source code, and then build and analyze a special version of the software. In addition to being time-consuming, this approach was useless for embedded designs. An embedded design's real-time I/O characteristics cannot be meaningfully simulated on the development computer system.

With iPAT software analysis, the engineer can quickly and easily show real execution time for program modules, procedures, lines, or user-defined symbolic ranges in the actual 80286 embedded or reprogrammable system environment. Software analysis with iPAT-286 provides histograms and tabular displays with the high-level names for program partitions, such as modules and procedures. iPAT automatically builds an internal data structure for correspondence of symbols and addresses using 8086 or 80286 object module format information (OMF86 and OMF286). This information is incorporated in executable files built with Intel software tools. (For object code built with software tools from other vendors, conversion is possible using utilities supplied with the iPAT host software package.)

Software analysis with iPAT is non-intrusive and carried out in real-time. iPAT does not sample program execution to build a statistical model; rather, it collects and compares all addresses in real-time.

Software teams coding in assembly language to increase program speed will now find they can change their methods to be more productive. They can write applications in high-level languages for easier expression of complex algorithms and easier code maintenance. Many parts of an application may have no significant impact on overall execution speed. Those parts that do can be isolated through iPAT software analysis. The algorithms can then be improved, or the code converted into special-purpose assembly language routines to improve execution speed.

Optimizing Hardware Interactions

iPAT-286 also enables the developer to examine the timing characteristics of interactions between hard-

ware devices and application software. By this means the developer can measure real-time characteristics that are critical to embedded applications, such as the latency between detecting a sensor condition for an aircraft and executing software routines that respond to the condition.

Ensuring Reliability

To ensure application reliability, developers typically devise unit-level and system-level test suites. A theoretical understanding of the design is used to generate test suite data. Then the application is executed and its output is examined. Depending on application complexity and testing time, a higher or lower statistical confidence level for the thoroughness of testing is obtained.

'Black box' testing of this sort can only evaluate inputs and outputs; not the actual execution of application code in real-time, which remains a matter of inference. Thus, in the past, evaluators could not be confident that their test cases actually executed all the code. Errors in untested code could not be detected because it was not known that the code was not executed by the test cases. Many developers were forced to trade off lengthening test times against the need to hit market windows, or to limit innovation.

With iPAT real-time code coverage analysis, the engineer can quickly and easily identify which lines, procedures, and modules are executed by test cases and which are not. Armed with this information, the evaluator determines incremental test input to hit the unexecuted portions of the application. The result is more efficient and effective testing, and greater freedom to innovate without sacrificing quality.

Impact of Real-Time Software Analysis on Development

Developers must introduce microprocessor-based products into competitive environments and product acceptance processes where performance and reliability are critical to success. iPAT provides a means of controlling performance and reliability to meet design expectations and a higher level of productivity in several phases of the product cycle.

In addition to supporting initial product development, iPAT can be used during the maintenance cycle to increase the quality of enhancements and reduce their cost. For example, using iPAT code coverage analysis to execute existing test suites for evaluation of subsequent releases enables engineers to determine more quickly and easily the incremental test cases needed to produce full code coverage. Developers also can more easily isolate and correct the performance bottlenecks that frequently result from enhancements.

The front end of the development cycle also can be improved by iPAT's real-time software analysis. New product designs often re-use existing code modules to reduce development costs. If good performance and reliability parameters have been built in and are known from previous software analysis, then the work of the designer who re-uses modules will be more productive.

Finally, real-time software analysis encourages engineers to engage in 'what-if' thinking. They can ask, 'What if this portion of the application were designed this way, or used this algorithm?' Alternatives can be compared to determine which is most efficient in the actual application environment.

iPAT™-286 OPERATION

Gaining real-time software analysis information in symbolic terms is simple with iPAT-286. The example here assumes an embedded 80286 protected-mode application, but applies similarly to applications that are loaded by an operating system or execute in real mode.

Obtaining Execution Histograms

To get started, the user replaces the 80286 chip in the target system with the iPAT-286 probe, supplies power to the iPAT system, and starts the 80286 embedded application. The next step is to load program symbolics into iPAT on the host PC system, instruct iPAT to profile, run the application in the target system, and display the results on iPAT. iPAT commands issued would be:

```
LOAD <my_prog.286>
PROFILE PROC INC :cdu

   /* profile procedures in the CDU
      module with inclusion of time
      in other subroutine calls */

   /* run application in target
      system */

DISPLAY
```

The default display is a histogram (see Fig. 1), ordered from low to high addresses. Histogram bars indicate relative time consumption. The cumulative execution times for each procedure also are shown.

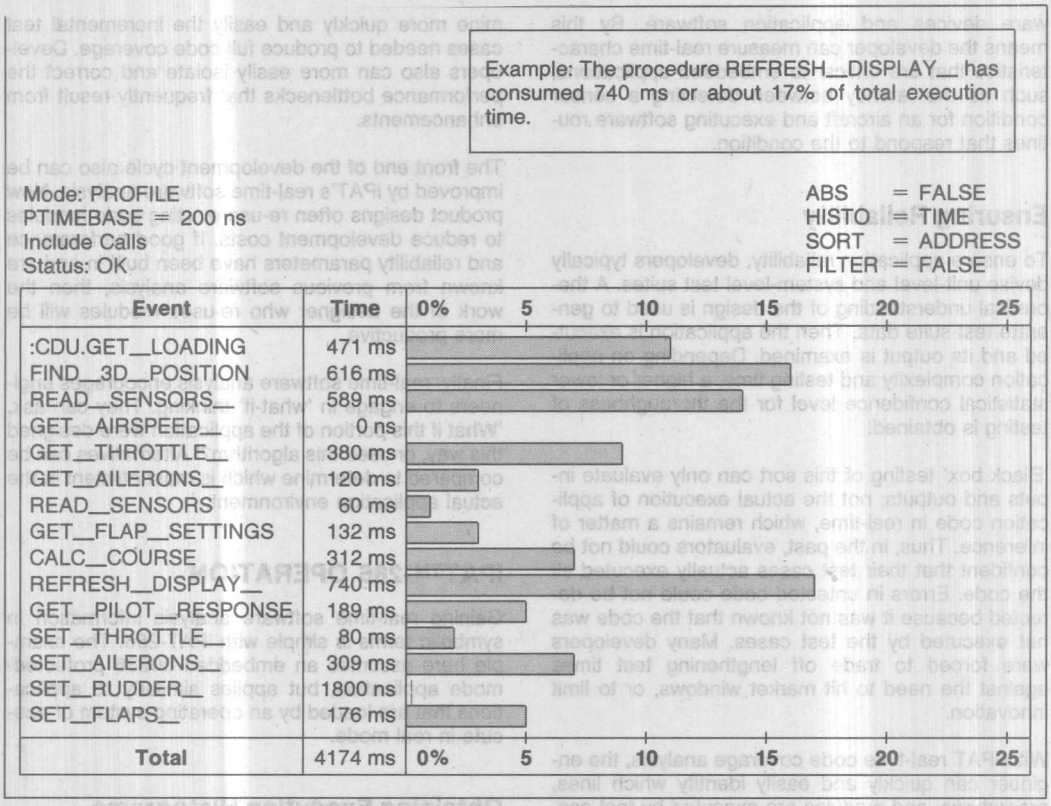


Figure 1. Histogram with Include Calls

Information provided at the top left of the screen indicates profiling at a granularity of ± 200 nanoseconds with inclusion of any time spent in called subroutines. Information provided at the top right indicates further display options (default values are shown). For example, the information is sorted in order of ascending addresses, there is no filtering of what is shown in the display, and 'background' time is not included (ABS = FALSE).

Displays have default settings but are flexible. For example, the user can change the display to show descending order by relative time consumption and to show only those procedures consuming more than 600 milliseconds. Instead of displaying results once, typing DISPLAY UPDATE 0 would produce a continuously updated display:

```
SORT = TIME
FILTER = TIME > 600 MS
DISPLAY UPDATE 0
```

The display now continuously refreshes with information only for those procedures that have accumulated more than 600 milliseconds of execution time.

Greater detail is obtained (see Fig. 2) by deselecting the histogram form of display:

```
HISTO = FALSE
DISPLAY
```

Example: GET__THROTTLE__ was executed 49 times. Total execution time was 380 ms with 7755 μs as the mean, 2002 μs as the minimum, and 16 ms as the maximum timing over the 49 executions.

Mode: PROFILE
PTIMEBASE = 200 ns
Include Calls
Status: OK

ABS = FALSE
HISTO = FALSE
SORT = ADDRESS
FILTER = FALSE

Event	Count	Time	Time Min	Time Avg	Time Max
:CDU.GET__LOADING	3	471 ms	157 ms	157 ms	157 ms
FIND__3D__POSITION	14	616 ms	14 ms	44 ms	181 ms
READ__SENSORS__	31	589 ms	7000 μs	19 ms	21 ms
GET__AIRSPEED__	0	0 ns	0 ns	0 ns	0 ns
GET__THROTTLE__	49	380 ms	2002 μs	7755 μs	16 ms
GET__AILERONS__	26	120 ms	1110 μs	4615 μs	11 ms
READ__SENSORS	14	60 ms	1000 μs	4285 μs	9020 μs
GET__FLAP__SETTINGS	12	132 ms	9000 μs	11 ms	34 ms
CALC__COURSE	26	312 ms	7000 μs	12 ms	14 ms
REFRESH__DISPLAY__	2	740 ms	72 ms	370 ms	668 μs
GET__PILOT__RESPONSE	3	189 ms	63 ms	63 ms	63 ms
SET__THROTTLE__	2	80 ms	40 ms	40 ms	40 ms
SET__AILERONS__	3	309 ms	103 ms	103 ms	103 ms
SET__RUDDER__	1	1800 ns	1800 ns	1800 ns	1800 ns
SET__FLAPS__	11	176 ms	14 ms	16 ms	17 ms
Totals	197	4174 ms			

Figure 2. Profile Include Calls Tabular Display

Examining Function Timing

Most applications have specifications for the timing of certain functions, such as worst-case timing to update displays, response to hardware interrupts, or transition timing characteristics. These types of timing can be characterized with specialized iPAT analysis modes.

In the following example, the DURATION mode is used to obtain a distribution of the time required to refresh a cockpit display screen. This function is initiated in the CDU module at line #145 and completed with a call to the BIOS module's WRITE__TTY function. For convenience, the engineer can define a range name for this part of the program's functional flow. The engineer can then set up iPAT to characterize the duration of this range. In the example (see Fig. 3), logarithmic scaling of timing bins from 200 nanoseconds to 1 second is specified:

```
DEFINE RANGE tty_update_time =
          :CDU#145 TO :BIOS.WRITE_TTY
DURATION tty_update_time LOGB 200 NS
                                TO 1SEC

   /* run application */
```

The distribution peak shows that, generally, execution was very fast, but a second peak and outliers are evident. These may be exceeding performance requirements.

Mode: DURATION
Event: TTY__UPDATE__TIME
Time Range: 200 ns to 1 sec
PTIMEBASE = 200 ns
Status: OK

SELECT = INNER
HISTO = COUNT

Type: LOGB

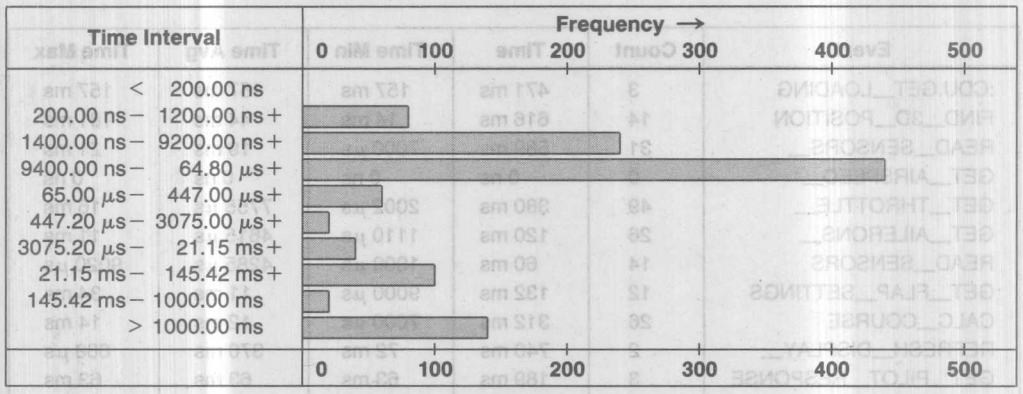


Figure 3. Duration of TTY__UPDATE__TIME

Other performance requirements for this application might define worst-case latencies to various hardware interrupts. For example, worst-case latency of no more than 256 microseconds from a hardware interrupt issued at ground contact by the landing gear until code is executed to shut down the aircraft's jet engines. Using the LATENCY analysis mode of iPAT, interrupt latency can be characterized. The example (see Fig. 4) uses logarithmic scaling of timing bins from 1000 nanoseconds to 256 microseconds:

```
LATENCY interrupt TRANS
:eng_mgr.stop_thrust_
LOGB 1000 NS TO 256 US

 /* attach iPAT-286 interrupt clip to
    device issuing interrupt and run
    application using developer's test
    harness */
```

Typical latency is slightly less than 1 μs but in about 13 cases exceeded the worst-case requirement of 256 μs.

Mode: LATENCY
Event: INTERRUPT TRANS :ENG_MGR.STOP_THRUST_
Time Range: 1 μs to 256 μs
PTIMEBASE = 200 ns
Status: OK

SELECT = INNER
HISTO = COUNT
Type: LOGB

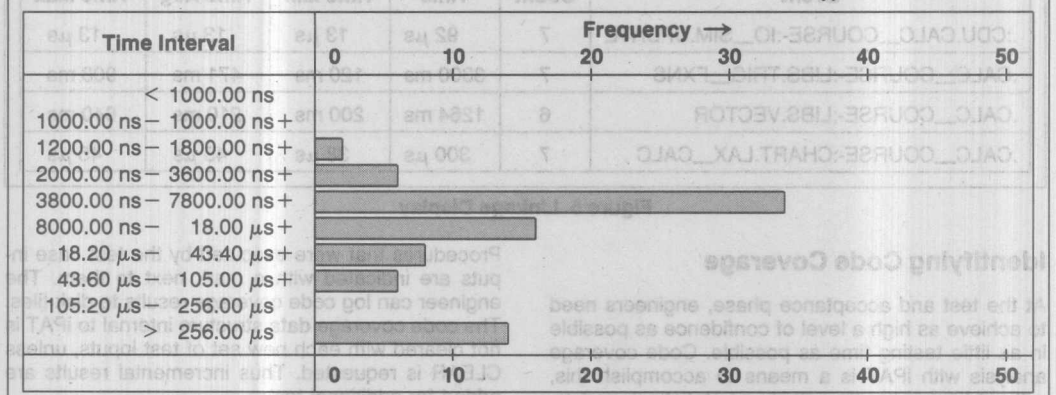


Figure 4. Latency

In many applications timing requirements also exist for the linkage between certain functional parts of the program. The LINKAGE mode of iPAT measures procedure interaction of this type. In the example (see Fig. 5), linkage between a procedure that calculates flight course corrections and several other program functions is characterized:

```
LINKAGE :cdu.calc_course LINK :io_sim.update, &
        :cdu.calc_course LINK :libs.trig_fxns, &
        :cdu.calc_course LINK :libs.vector, &
        :cdu.calc_course LINK :chart.lax_calc
```

Timing data suggests that the interaction of the CALC__COURSE routine with the TRIG__ FXNS is a likely source of bottlenecks.

Mode: LINKAGE
PTIMEBASE = 200 ns
Status: OK

Event	Count	Time	Time Min	Time Avg	Time Max
.:CDU.CALC__COURSE-:IO__SIM.UPDATE	7	92 μs	13 μs	13 μs	13 μs
.CALC__COURSE-:LIBS.TRIG__FXNS	7	3300 ms	120 ms	471 ms	908 ms
.CALC__COURSE-:LIBS.VECTOR	6	1264 ms	200 ms	210 ms	640 ms
.CALC__COURSE-:CHART.LAX__CALC	7	300 μs	38 μs	43 μs	45 μs

Figure 5. Linkage Display

Identifying Code Coverage

At the test and acceptance phase, engineers need to achieve as high a level of confidence as possible in as little testing time as possible. Code coverage analysis with iPAT is a means to accomplish this, without having to rely entirely on statistical models. COVERAGE mode identifies parts of the code that have been executed by tests and those that have not. Incremental testing inputs then can be determined, or sections of 'dead code' removed from the program. Code coverage data also provides a means for the evaluator to document testing results.

In the example (see Fig. 6), code coverage results are shown at procedure level. The commands to obtain these results were:

```
COVERAGE :cdu

   /* now execute test cases on the
      application system. /*

DISPLAY
```

Procedures that were executed by the test case inputs are indicated with a mark next to them. The engineer can log code coverage results to disk files. The code coverage data structure internal to iPAT is not cleared with each new set of test inputs, unless CLEAR is requested. Thus incremental results are added for additional tests.

The upper right part of the display indicates that procedure-level results are shown. This is the default. Without re-executing tests, the display can be changed to line-level or module-level results. All modules, procedures, and lines can be displayed. The only limitation is that all areas specified for code coverage must comprise a total of no more than 252K of program memory, which need not be contiguous. Memory space(s) to be covered are automatically programmed by iPAT from the user's symbolic specifications, such as COVERAGE :mod__a to :mod__n,:mod__p, :mod__u.

Utilities for producing address-by-address boolean maps of code coverage or instruction-level output are also provided with iPAT.

The procedure READ_SENSORS_ is marked as having been executed; GET_AIRSPEED_ is not. 80% of the procedures were executed. Note that this data can be redisplayed at the line level.

Mode: COVERAGE SHOW = ALL PROC

Exec	Event	Exec	Event	Exec	Event
:CDU					
■	GET_LOADING	■	GET_AILERONS_	■	GET_PILOT_RESPONSE
■	FIND_3D_POSITION	■	READ_SENSORS		SET_THROTTLE_
■	READ_SENSORS_	■	GET_FLAP_SETTINGS	■	SET_AILERONS_
	GET_AIRSPEED_	■	CALC_COURSE		SET_RUDDER_
■	GET_THROTTLE_	■	REFRESH_DISPLAY_	■	SET_FLAPS_

EXEC: 12 (80.0%) NOEXEC: 3 (20.0%) Total: 15

Figure 6. Coverage Display

USAGE WITH AN IN-CIRCUIT EMULATOR

iPAT-286 monitors real-time software activity symbolically and passively to provide the software engineer with an instrument analogous to an oscilloscope for the hardware engineer. But in some development environments, interaction with the prototype is needed: for example, downloading code or writing data to an incomplete prototype. iPAT-286 has the flexibility required for use in conjunction with an Intel 80286 emulator system. Under this scenario, the emulator and iPAT have separate PC hosts, but share the in-circuit emulator's 80286 probe. By this means, unit-level tests can be generated procedurally by the in-circuit emulator and analyzed by the iPAT system.

HOST SYSTEM REQUIREMENTS

An IBM PC AT or PC XT system running PC-DOS 3.xx with at least 512K bytes of available memory in RAM.

An RS232 serial port that operates at 300, 1200, 9600 (default), or 19200 baud must be available on the host system.

ANALYSIS MODES

Modes with Multiple Partitions

The following iPAT modes collect execution information for multiple modules, procedures, lines, user-defined ranges, or absolute address partitions.

Profile mode: Collects time and count profiles for up to 125 program partitions (modules, procedures, line numbers, or user-defined ranges). Time information includes absolute cumulative, minimum, maximum, and average execution time for each partition. Profile information can be collected using any of three options: INC, NOINC, or EXC. These options set profile mode to include calls, not include calls, or show range activity under execution that is broken up by interrupts and called routines. The user can display information in histogram or tabular form, and sort by address, cumulative time, or cumulative count. Filters can be set to select a subset of partitions that have specified characteristics.

Linkage mode: Collects time and count information for the linkage interaction of up to 63 events, such

as procedure to procedure activity, including recursive activity. The user can display information in histogram or tabular form, and sort by address, cumulative time, or cumulative count.

Usage mode: Collects time and count information about usage of one 'used' event by up to 63 other 'user' events, such as a subroutine that is called by many subroutines. The user can display information in either histogram or tabular form, and sort by address, cumulative time, or cumulative count.

Coverage mode: Provides coverage information for up to 63 4K-byte blocks of address space, which need not be contiguous and are mappable anywhere in the 80286 physical address space. Coverage areas are set automatically by specifying partitions for coverage with module, procedure, line number, or absolute address boundaries. Boolean coverage maps are automatically converted into partition names such as modules, procedures, line numbers, or user-defined ranges by the iPAT host software for user display. Coverage information is cumulative, incremental, and can be recorded to a user-specified file, or recalled, by using iPAT's SAVE and RECALL commands.

Modes With Multiple Timing Bins

The following iPAT modes organize data into timing bins for a single partition. The partition can be a procedure or starting-ending pair of events bounding the partition). Users can select bin granularity (4, 8, 16, or 32 bins), linear or logarithmic scaling, as well as the lower and upper bounds for the range of bins. The distribution of execution times can be displayed as a histogram or as a table, including minimum, maximum, and cumulative execution times.

Duration mode: Collects a distribution of execution times across a range of timing bins for a single partition.

Latency mode: Collects a distribution of latency from a hardware interrupt to a software execution event, or between interrupts, across a range of timing bins. A clip line from the iPAT probe to the source of the interrupt signal is used to inform iPAT of asynchronous events external to the software execution address stream.

Windowed-event count mode: Collects count data concerning an event that occurs within a specified event window.

FUNCTIONALITY

Partition specification: Program partitions can be specified using program symbolic names, user-defined symbolic names, or absolute addresses. Symbols anywhere in the 80286 physical or virtual address space can be used. Execution, however, is monitored using events in the physical address space of the 80286.

Symbolic Information: Program symbolics for modules, procedures, and line numbers are loaded by iPAT host software using OMF86 or OMF286 (object module formats) in absolute, load-time locatable, or single-task loadable files built with Intel 86 or 286 compilers using the DEBUG option and Intel linkers, locaters, builders, and binders. Utilities are included for converting non-Intel symbolic information to iPAT-loadable symbolics.

Speed: The iPAT-286 probe supports the 80286 to 12.5 MHz and collects instruction addresses at full processor speeds (however, sustained execution in a short loop may cause iPAT processing overflow).

Timebase: The iPAT data collection timebase is selectable at 200 nanoseconds, 1 microsecond, 10 microseconds, or 100 microseconds.

Display updates: An update frequency can be specified by the user at an interval in seconds.

Status: If time-count, bin-count, or FIFO stack overflow occurs, iPAT displays indicate the error status.

Session logs: The user can log part or all of an iPAT session, including displays and input/output in the iPAT environment, by using the LIST command to initiate recording to a specified log file and the NOLIST command to end recording.

Command input: All commands can be issued from iPAT's context-sensitive command line, which prompts the user with valid options. Commonly used commands also can be issued through predefined function keys. The user may define additional function keys and automatically include these in future sessions.

iPAT definitions: iPAT provides a powerful set of procedural constructs to bring greater flexibility and productivity to the engineer's software analysis. For example, partition specifications and analysis controls can be collected in a 'SETUP' object and then

edited, saved, and recalled. Display controls can be defined as a TEMPLATE. LITERALLY and RANGE definitions can be created, saved, and re-used for a symbolic short-hand to abbreviate frequently repeated commands and to add high-level range names during analysis.

UTILITIES

Various utilities are supplied with the iPAT host software. These provide, for example, conversion of input files to a form offering symbolic functionality when loaded into iPAT, and conversion of output files to a form in which instruction-level coverage information can be generated by the user.

Microsoft* symbolics: MS*-LINK map file address information for modules, procedures, and lines may be converted to an OMF86-compatible, iPAT-loadable symbolics file.

Other non-OMF symbolics: Address/symbolic information for module, procedure, and lines generated by the user from other vendors' tools in a standard text file format also can be converted to OMF86-compatible, iPAT-loadable symbolics. After loading the converted information into iPAT, the user must be able to supply the bases to which code segments are loaded.

Boolean coverage map: A boolean map indicating all addresses covered and not covered during execution can be produced by saving iPAT code COVERAGE results to a user-specified file and then processing the file through a utility.

Instruction-level coverage: An instruction-level coverage text file can be produced by processing a boolean map (see above) of address execution and a user-supplied file containing disassembly of the code memory image through a utility supplied with the iPAT-286 software. The disassembly file is a standard formatted text file produced, for example, by use of a software debugger or an in-circuit emulator from the memory image of the executable code.

SPECIFICATIONS

Physical Characteristics

iPAT-286 PROBE:

Length: 15.3 cm (6 in)
Width: 10.2 cm (4 in)
Height: 2.86 cm (1.125 in)

iPAT CORE MODULE:

Length: 35 cm (13¾ in)
Width: 21 cm (8¼ in)
Height: 4.5 cm (1¾ in)

iPAT POWER SUPPLY:

Length: 28 cm (11 in)
Width: 20 cm (7¾ in)
Height: 11 cm (4¼ in)

AC power cord for the power supply: 3.0 m (10 ft)

Power-supply-to-core DC power cable: 1.8 m (6 ft), 10 conductor

iPAT probe-to-target cable: 10.2 cm (4 in)

iPAT probe-to-core interface cable: 0.9 m (36 in), 60 conductor

RS232 serial cables for connecting iPAT core to PC XT or PC AT host system: 3.7 m (12 ft)

Electrical Characteristics

Selectable AC power source: 100V, 120V, 220V, 240V

47 Hz–63 Hz

2 amps (AC) at 100V or 120V; 1 amp at 220V or 240V

Environmental Requirements

Operating temperature:	10°C to 40°C (50°F to 104°F)
Operating humidity:	Maximum of 85% relative humidity, non-condensing

ORDERING INFORMATION

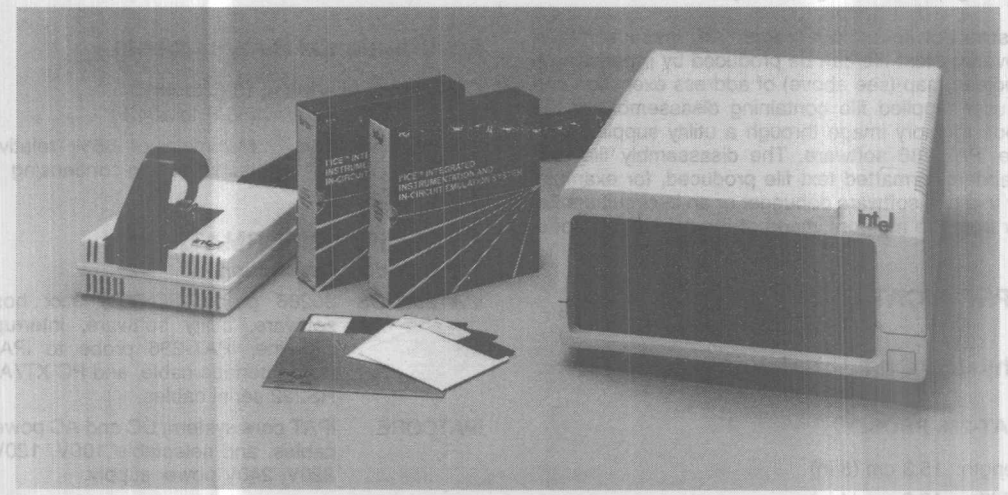
Order Code	Description
iPAT286DOS	80286 probe, PC-DOS 3.xx host software, utility software, interrupt clip line, iPAT-286 probe to iPAT Core interface cable, and PC XT/AT RS232 serial cables.
iPATCORE	iPAT core system, DC and AC power cables, and selectable 100V, 120V, 220V, 240V power supply.

*MS-LINK, Microsoft are trademarks of Microsoft Corp.

I²ICE™ Integrated Instrumentation and In-Circuit Emulation System

- **Provides Real-Time In-Circuit Emulation**
- **Offers Symbolic Debugging Capabilities**
  - **— Accesses Memory Locations and Program Variables (Including Dynamic Variables) Using Program-Defined Names**
  - **— Maintains a Virtual Symbol Table**
  - **— Source Code Display at Breakpoints**
- **Offers Multi-Condition, Multi-Level, Multi-Probe Break and Trace Capability**
- **Provides Built-In AEDIT Editor to Allow Editing of Development System Files without Exiting from I²ICE Operation**
- **Provides Low Cost Conversions Among 8086, 8088, 80186, 80188 and 80286 Microprocessors**
- **Simultaneously Controls up to Four Microprocessors for Debugging Multiprocessor Systems for a Single Work Station**
- **Supports Common Memory between Processors without Any User System Hardware**
- **Offers a Performance Analysis Tool (iPAT™ Analyst)**
- **Maps User Program Memory into a Maximum of 288K Zero-Wait-State RAM (Zero Wait-States to 10 MHz)**
- **Maps User I/O to Console or to Debugging Procedures**
- **Provides Disassembly and Single-Line Assembly to Help with On-Line Code Patching**
- **Common Human Interface Provided by the PSCOPE-86 Debugging Language and the I²ICE Command Language**
- **Uses Integrated Command Directory, ICD™, for Command Syntax Direction/ Correction to Ease Debug Operations**

The Intel Integrated Instrumentation and In-Circuit Emulation (I²ICE™) system aids the design of systems that use the 8086, 8088, 80186, 80188 and 80286 microprocessors. The I²ICE system combines symbolic software debugging, in-circuit emulation, and the optional Intel Performance Analysis Tool (iPAT analyst). Support features for the 8087 and 80287 coprocessors are also included. For the 8086/8088, 80186/80188, and 80286 processors, the I²ICE system supports programs written in C, PL/M, FORTRAN, Pascal, and assembly language. Up to four I²ICE instrumentation chassis can be hosted by one of Intel's Intellec® microcomputer development systems or by an IBM* PC AT or PC XT.



210469-1

*IBM is a trademark of the International Business Machines Corporation.

October 1987
Order Number: 210469-008

PHYSICAL DESCRIPTION

The I²ICE system hardware consists of the host interface board, the I²ICE instrumentation chassis, the emulation base module, the emulation personality module, a host/chassis cable, inter-chassis cables (for multiple chassis systems), a user cable, optional high-speed memory boards, and an optional performance analyzer. The I²ICE system software consists of I²ICE host software, I²ICE probe software, confidence tests, PSCOPE-86, and optional iPAT analyst software. Table 1 shows elements of the I²ICE system.

The host interface board resides in the host development system. A cable connects the host interface board to the I²ICE instrumentation chassis. Another cable connects the I²ICE instrumentation chassis to the buffer box.

The instrumentation chassis contains high-speed zero-wait-state emulation memory, break-and-trace logic, memory and I/O maps, and the emulation clips assembly.

The chassis may also contain the optional performance analyzer and optional high-speed memory. High-speed memory is expandable from 32K bytes to 288K bytes in 128K increments.

The buffer box contains the emulation personality module. This module configures the I²ICE system for a particular iAPX microprocessor. The user cable connects the buffer box to user prototype hardware.

The host development system may host up to four I²ICE instrumentation chassis. Each chassis may have its own buffer box, user cable, emulation clips, optional high-speed memory boards, and performance analyzer.

TARGET SYSTEM CONSIDERATIONS

To ensure proper emulation of a host target system, consider the following:

- Each I²ICE system probe has specific timing parameters that differ from the chip which the probe emulates. Hence, a customer design that follows the chip's timing specifications may not work with the I²ICE system probe. The target system may have to be modified slightly to account for the differences in timing between the probe and the chip. See the probe waveform section in this data sheet for timing differences.
- Target system noise and signal margins and timings are a critical consideration for emulation at speeds beyond 6 MHz. Typical solutions used to reduce target system noise such as RC networks and series resistor terminations could cause unacceptable timing degradation. Consequently, Intel recommends that wirewrap target boards be carefully designed for emulation with the I²ICE system. Printed circuit boards should be used because of the superior signal transmission characteristics. All target systems must have power and ground planes, decoupling capacitors, and signal lines layed out according to correct design techniques. For an introduction to proper design, see Application Note 125, Designing Microcontroller Systems for Electrically Noisy Environments, Order Number 210313.
- The I²ICE system depends on a target system clock signal to run the internal probe circuitry. To run the internal probe circuitry, the clock signal must satisfy two criteria. The target system clock must meet the voltage levels defined in this data sheet and it must also exceed the TTL logic family minimal noise and ringing specifications. This is necessary since the signal must travel up the user cable and through data buffers to reach the probe circuitry. The I²ICE system is designed to minimize the capacitive, noise, and chip delay associated with this path, but these effects worsen timings and amplify target system noise that may exist.

FUNCTIONAL DESCRIPTION

Resource Borrowing

The I²ICE system memory map allows the prototype system to borrow memory resources from the I²ICE system.

If prototype memory is not yet available, the user program may reside in I²ICE memory. Because this memory is RAM, changes can be made quickly and easily. For example, if the prototype contains EPROM, it does not need to be erased and reprogrammed during development.

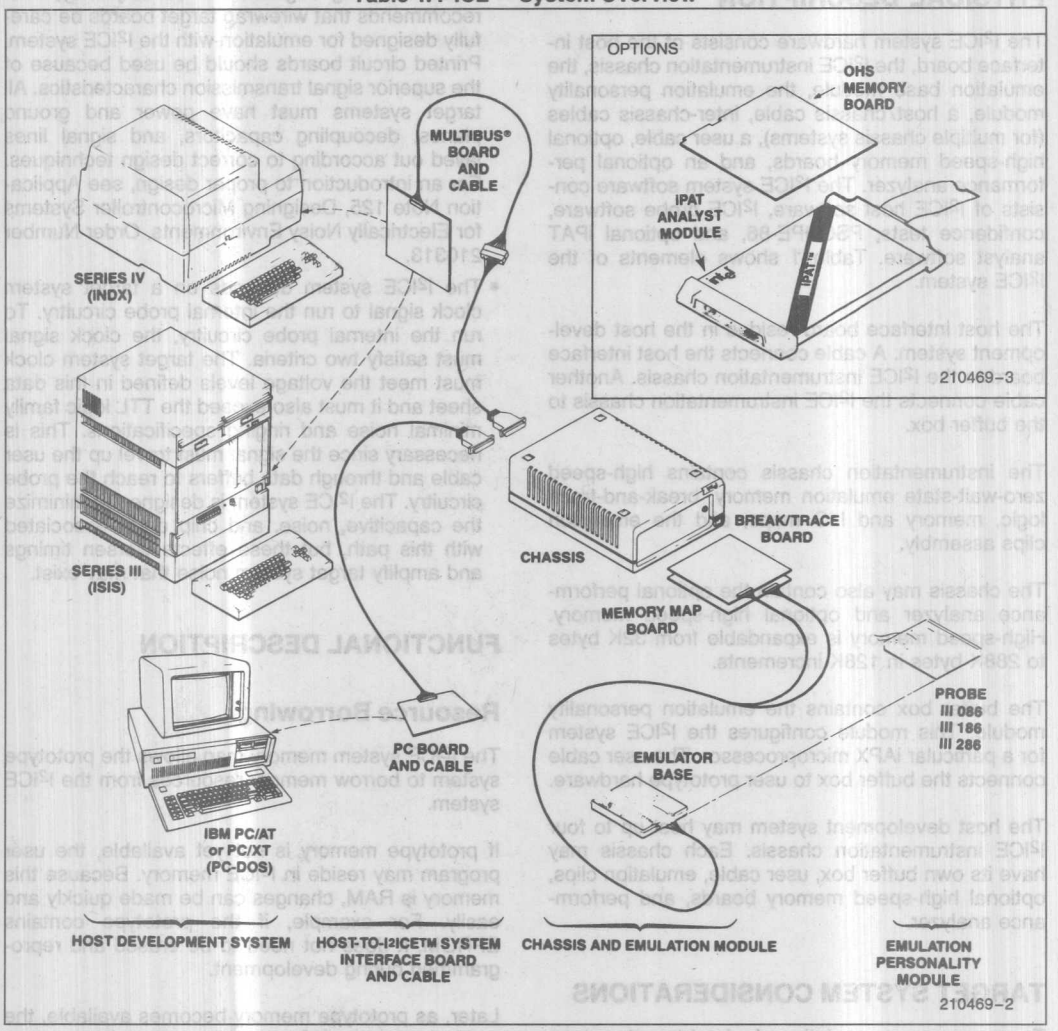
Later, as prototype memory becomes available, the verified user program can be reassigned, memory block by memory block, to prototype memory.

The I²ICE™ System Memory Map

The I²ICE system can direct (map) an emulated microprocessor's memory space (the user program memory) to any combination of the following:

- High-speed I²ICE system memory—this consists of 32K bytes of programamble wait-state memory (programmable from 0 to 15). This memory resides in the I²ICE system chassis on the map-I/O board.

Table 1. I²ICE™ System Overview



Name	Description
Host Development System	Required for all applications. Use one of the following: • Intellec Series III development system • Intellec Series IV development system • IBM PC AT or PC XT (with 512K bytes of available memory and version 3.0 of PC DOS) • IBM 50 system (available in Japan; features *kanji*)

Table 1. I²ICE™ System Overview (Continued)

Name	Description
Host-to-I²ICE System Interface Board, Cable, and Host Software	Required for communication between the host and the I²ICE system. • MULTIBUS® bus interface board for Series III and Series IV (product code III520) • Host-to-I²ICE system cable for the Series III and Series IV (product code III530 or III531) • I²ICE system host software for the Series III and Series IV (product code III951A, B, or C) • Package with PC host interface board, cable and PC DOS version of I²ICE host software (product code III520AT954D)
Instrumentation Chassis and Emulation Module	Required for real-time microprocessor emulation, break and trace capability, and memory and I/O capability. • Instrumentation chassis (product code III514B) has four board slots: 1 slot for break/trace board 1 slot for map-I/O board 2 slots for 1 (or 2) optional high-speed memory board(s) and/or 1 optional logic timing analyzer board • Maximum of four chassis for multi-probe applications • Emulation module (product code III620) includes break/trace board, map-I/O board, and buffer base box
Emulation Personality Module (Probe) and Probe Software	Required for emulation of specific microprocessors: 8086/8088, 80186/80188, or 80286. • Module includes personality board, buffer box cover, and user cable • Series III or IV: Order probe and probe software separately • PC host: Probe and probe software packaged together
Intel Performance Analysis Tool (iPAT Analyst)	Used to optimize code execution speed and control and to improve software reliability. • Complete with sytem software, power supply, core module, iPAT-to-I²ICE interface board, and cables
Optional High-Speed Memory Board (OHS)	Required for memory expansion. • 128K bytes of programmable (0 to 15) wait-state memory • One or two boards mount in the instrumentation chassis

- Optional high-speed I²ICE memory—this consists of up to 256K bytes of programmable wait-state memory (0 wait-states up to 10 MHz). This memory resides in the I²ICE system chassis on one or two optional high-speed memory boards (128K bytes each).
- MULTIBUS® bus memory (host system memory)—this resides in the host development system itself. (Any amount of unused host memory can be used in 1K increments.) Note that this feature is not available for a PC host.
- User memory—this resides in the user prototype hardware.

When a user program runs in I²ICE memory or user memory, the I²ICE system emulates in real time. A memory access to MULTIBUS bus memory, however, inserts approximately 25 wait-states into the memory cycle.

Access Restrictions

In addition to directing memory accesses, the following access restrictions can be specified.

- Read-only—the I²ICE system displays an error message if a user program attempts to write to an area of memory designated as read-only. The user can, however, write to a read-only area with I²ICE system commands.
- Read/write, no verify—normally, the I²ICE system performs a read-after-write verification after program loads and after writing to memory with an I²ICE system command. The I²ICE system can suppress this verification. For example, if a prototype has memory-mapped I/O, a verifying read may change the state of the I/O device.
- Guarded—initially, the I²ICE system puts all memory in a guarded state. Neither the user program nor the I²ICE system user can access guarded memory.

The I²ICE™ System I/O Map

The I²ICE system can direct (map) an emulated microprocessor's I/O space to the host development system's console, to the prototype system, to debugging procedures, or to a combination of these.

SIMULATING I/O WITH THE HOST DEVELOPMENT CONSOLE

Suppose a user program requires input from an I/O device not yet part of the prototype. Map the input port range assigned to that device to the host development system's console. Then, when the user program requires input, it halts and the I²ICE system console displays a message requesting the data. When you enter the required data at the keyboard, the user program continues.

SIMULATING I/O WITH I²ICE™ SYSTEM DEBUGGING PROCEDURES

Procedures that supply the needed input data can be written in the I²ICE system command language. When setting up the I/O map, the user specifies that the I/O procedure is invoked when certain I/O ports are accessed.

I/O ports are mapped in blocks of 64 byte-wide ports or 32 word-wide ports. A total of 64K byte-wide ports or 32K word-wide ports can be mapped.

Symbolic Debugging

With symbolic debugging, a memory location can be referenced by specifying its symbolic reference. A symbolic reference is a procedure name, line number, or label in the user program that corresponds to a location in the user program's memory space.

TYPICAL SYMBOLIC FUNCTIONS

Symbolic functions include:

- Changing or inspecting the value and type of a program variable by using its program-defined name, rather than the address of the memory location where the variable and a hexadecimal value for the data are stored.
- Defining break and trace events using source-code symbols.

With symbolic debugging, the user can reference static variables, dynamic (stack-resident) variables, based variables, and record structures combining primitive data types. The primitive data types are ADDRESS, BOOLEAN, BYTE, BCD, CHAR, WORD, DWORD, SELECTOR, POINTER, three INTEGER types, and four REAL types.

THE VIRTUAL SYMBOL TABLE

The I²ICE system maintains a virtual symbol table for program symbols; that is, the entire symbol table need not fit into memory at the same time. (The size of the virtual symbol table is constrained only by the capacity of the storage device.)

The I²ICE system divides the symbol table into pages. If a program's symbol table is large, the I²ICE system reads only some of the symbol table pages into memory. When the user references a variable whose symbol is not currently defined in memory, the I²ICE system reads the needed symbol table page from disk into memory.

Breakpoint, Trace, and Arm Specifications

With I²ICE commands, breakpoint, trace and arm specifications can be defined.

Breakpoints allow halting of a user program in order to examine the effect of the program's execution on the prototype. With the I²ICE system, a breakpoint can be set at a particular memory location or at a particular statement in a user program (including high-level language programs). A break can also be set to occur when the user program enters or accesses a specified memory partition or reads or writes a user program variable. When the user program resumes execution, it picks up from where it left off.

Normally, the I²ICE system traces while the user program executes. With a trace specification, however, the user can choose to have tracing occur only when specific conditions are met.

An arm specification describes an event or combination of events that must occur before the I²ICE system can recognize certain breakpoint and trace specifications. Typical events are the execution of an instruction or the modification of a data value.

The I²ICE system command language allows you to specify complex, multilevel events. For example, you can specify that a break occurs when a variable is written, but only if that write occurs within a certain procedure. The execution of the procedure is the arm condition; the variable modification is the break condition. The I²ICE system command language allows users to specify complex events with up to four states with four conditions and to use such events as arm, break, or trace conditions; a specified number of events can be used as a condition.

SOURCE DISPLAY

With the source display commands, a user can correlate a module under debug to a source code file. Then, when breakpoints are encountered, source text is displayed along with the break message and the line number of the breakpoint. The number of source code lines displayed before and after a breakpoint can also be defined.

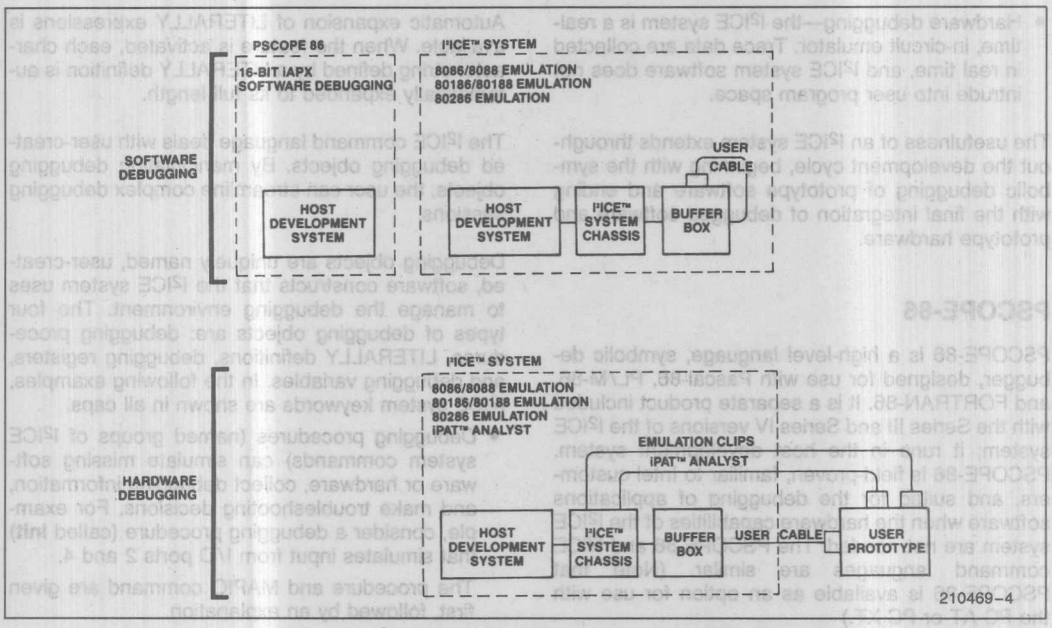


Figure 1. I²ICE™ System Debugging Capabilities

Coprocessor Support

The 8086/8088 emulation personality module provides transparent RQ/GT and MN/MX pin emulation to support real-time prototype systems that use the 8087 as a coprocessor. The 8086/8088 (and the 80186/80188) emulation personality module also provides debugging features specific to the 8087. I²ICE system commands provide access to the 8087's stack, status registers, and flags. The I²ICE system's disassembly and trace features extend to 8087 instructions and data types.

The 80186 and 80286 emulation personality modules also allow the prototype hardware to contain coprocessors. The 80186 probe can qualify break points and collect trace information when the coprocessor drives the status lines ($\overline{S0}$–$\overline{S2}$) in the prescribed manner. The 80286 personality module allows the hardware to contain the 80287 processor extension and provides special debugging features—the user can enable and disable the 80287 and change and examine its registers.

DEBUGGING WITH THE I²ICE™ SYSTEM

The I²ICE system allows both hardware and software debugging (see Figure 1).

- Software debugging—I²ICE system commands permit symbolic debugging of user programs written in high-level languages as well as assembly language. By looping the user cable back into the buffer box, a user program can be debugged even if no prototype hardware is present. In a multi-probe environment, the I²ICE system can map common memory from the host development system and support semaphore operation even with no user system prototype hardware. This feature makes possible detailed debugging of multi-processor software before the hardware is available.

  Additionally, as code is being developed, preliminary analyses can be made with the optional iPAT analyst. You can also use the I²ICE system and the iPAT analyst to analyze program execution under different target system conditions. This can be accomplished by setting up target system conditions in the I²ICE system and running the program with the iPAT analyst activated.

- Hardware debugging—the I²ICE system is a real-time, in-circuit emulator. Trace data are collected in real time, and I²ICE system software does not intrude into user program space.

The usefulness of an I²ICE system extends throughout the development cycle, beginning with the symbolic debugging of prototype software and ending with the final integration of debugged software and prototype hardware.

PSCOPE-86

PSCOPE-86 is a high-level language, symbolic debugger, designed for use with Pascal-86, PL/M-86, and FORTRAN-86. It is a separate product included with the Series III and Series IV versions of the I²ICE system; it runs in the host development system. PSCOPE-86 is field-proven, familiar to Intel customers, and suited for the debugging of applications software when the hardware capabilities of the I²ICE system are not needed. The PSCOPE-86 and I²ICE command languages are similar. (Note that PSCOPE-86 is available as an option for use with the PC AT or PC XT.)

Designing a product that contains a microcomputer requires close coordination of hardware and software development. A typical design process takes advantage of both the I²ICE system and PSCOPE-86. Use PSCOPE-86 for debugging software before downloading the software into a target environment; use the I²ICE system for debugging and emulation of the target system.

THE I²ICE™ SYSTEM COMMAND LANGUAGE

The syntax of I²ICE system commands resembles that of a high-level language. The I²ICE system command langauge is versatile and powerful while remaining easy to learn and use.

The Integrated Command Directory (ICD™) assists users with command syntax.

- The ICD directory directs the user in choosing commands from a display on the bottom line of the screen. As commands are entered, the bottom line indicates syntax elements available for use in the commands.
- The ICD directory flags syntax errors. Syntax errors are flagged as they occur (rather than after the carriage return is pressed).
- The ICD directory provides on-line help with the HELP command.

Automatic expansion of LITERALLY expressions is available. When the feature is activated, each character string defined by a LITERALLY definition is automatically expanded to its full length.

The I²ICE command language deals with user-created debugging objects. By manipulating debugging objects, the user can streamline complex debugging sessions.

Debugging objects are uniquely named, user-created, software constructs that the I²ICE system uses to manage the debugging environment. The four types of debugging objects are: debugging procedures, LITERALLY definitions, debugging registers, and debugging variables. In the following examples, I²ICE system keywords are shown in all caps.

- Debugging procedures (named groups of I²ICE system commands) can simulate missing software or hardware, collect debugging information, and make troubleshooting decisions. For example, consider a debugging procedure (called **init**) that simulates input from I/O ports 2 and 4.

  The procedure and MAPIO command are given first, followed by an explanation.

```
*DEFINE PROCEDURE init = DO
.*IF %0==2 THEN
..*PORTDATA=100T
..*ELSE IF %0==4 THEN
...*PORTDATA=65T
...*END
..*END
.*END
*MAPIO 0 LENGTH 64K ICE (init)
```

  Whenever the MAPIO command maps I/O ports to an I²ICE system procedure, three parameters are made available to the procedure (even if the procedure does not use them): %0, %1, %2. The parameter %0 passes the port number; %1 passes a Boolean value that indicates whether read or write I/O activity will occur, and %2 passes a Boolean value that indicates whether the I/O is a byte-wide or a word-wide port. PORTDATA is a pseudo-variable that contains the actual port data. This procedure specifies that if port 2 is used, the procedure returns 100 (base ten); if, however, port 4 is used, the procedure returns 65 (base ten).

- LITERALLY definitions are shorthand names for previously defined character strings. LITERALLY definitions can save keystrokes and improve clarity. For example, here is the definition of a LITERALLY that saves keystrokes. This LITERALLY allows the user to type DEF for DEFINE.

```
*DEFINE LITERALLY DEF = "DEFINE"
```

These definitions may be saved to disk and autoreloaded. In addition, an automatic LITERALLY expansion feature can be turned on and off.

- Debugging registers are user-created, software registers that hold arm, breakpoint and trace specifications. The I²ICE system can be ordered to emulate the user program and specify one or more debugging registers. There is no need to reenter the specification for each emulation. For example here is the definition of a debugging register called **pay** that contains a trace specification. This example takes advantage of the previous LITERALLY definition.

```
*DEF TRCREG pay = :cmaker.payment
```

To emulate a user program and trace only during the procedure **payment,** specify the debugging register **pay** as part of the GO command.

```
*GO USING pay
```

- Debugging variables are user-created variables used with I²ICE system commands. For example, here is the definition of a debugging variable called **begin.** Its type is POINTER.

```
*DEFINE POINTER begin = 0020H:0006H
```

During a debugging session, the user can set the execution point to this pointer value by typing:

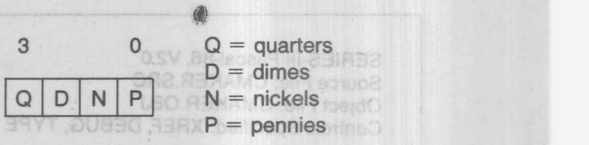
```
*$=begin
```

The I²ICE system pseudo-variable $ represents the current execution point.

Example of a Debugging Session

Figures 2, 3, and 4 illustrate some of the key capabilities of the I²ICE system. The user program is written in Pascal-86. It was compiled, linked, and located on an Intellec Series III development system. The resulting file consists of absolute code and is called CMAKER.86. Figure 2 shows the Pascal listing; Figure 3 shows a sample debugging session; and Figure 4 briefly explains the debugging steps shown in Figure 3.

The CMAKER.86 program controls an automatic changemaker. The program reads the amount tendered (the variable **paid**) and the amount of the purchase (the variable **purchase**). It calculates the coins needed for change and asserts control signals to a change release mechanism by writing an output port. Each of the lower four bits of the output port controls the release of a different coin denomination.



I²ICE™ System Command Functions

The I²ICE system command language contains a number of functional categories.

- Emulation commands—the GO command instructs the I²ICE system to begin emulation. The user can also command the I²ICE system to break or trace under certain specified conditions.
- Utility commands—these are general purpose commands for use in a debugging environment. For example, one use of the EVAL command is to calculate the nearest source-code line number that corresponds to the address of an assembly language instruction. The PRESRC command can be used to display a specified number of source code lines preceding a breakpoint. The HELP command provides on-line assistance. The EDIT command invokes a menu-driven text editor (AEDIT) that allows updating of debugging object definitions and editing of development system files without exiting the I²ICE system. The shell escape command () enables access to the DOS operating system without exiting the I²ICE system (DOS host specific). A command line editor and history key are also provided.
- Environment commands—these are commands that set up the debugging environment. For example, the MAP command sets up the memory map. Another environment command (WAITSTATE) inserts wait-states into memory accesses, allowing the simulation of slow memories.
- File handling commands—these are commands that access disk files. Debugging object definitions can be saved in a disk file and loaded in later debugging sessions. Debugging sessions can also be recorded in a disk file for later analysis.
- Probe-specific commands—these are commands whose effects are different for different probes. For example, the PINS command displays the state of selected signal lines on the current probe.
- Option-specific commands—these are commands that control an optional test/measurement device, such as the performance analysis tool.

```
SERIES-III Pascal-86, V2.0
Source File: CMAKER.SRC
Object File: CMAKER.OBJ
Controls Specified: XREF, DEBUG, TYPE

STMT   LINE   NESTING      SOURCE TEXT: MAKER.SRC
   1      1     0  0      PROGRAM cmaker;
   2      2     0  0      VAR change,coins                    :integer;
   3      3     0  0          quarters,nickels,dimes,pennies  :integer;
   4      4     0  0          paid,purchase                   :word;

   5      6     0  0      PROCEDURE payment;
   6      7     1  0        VAR numberofcoins                 :integer;
   7      8     1  0            release                       :word;
   8      9     1  0        BEGIN    (*payment*)
   8     10     1  1          numberofcoins:=quarters+dimes+nickels+pennies;
   9     11     1  1            while numberofcoins<>0 do
  10     12     1  1            BEGIN
  10     13     1  2            release:=0;
  11     14     1  2            if quarters<>0 then
  12     15     1  2                BEGIN
  12     16     1  3                release:=release+8;
  13     17     1  3                quarters:=quarters-1
                                  END;
  15     19     1  2            if dimes<>0 then
  16     20     1  2                BEGIN
  16     21     1  3                release:=release+4;
  17     22     1  3                dimes:=dimes-1
                                  END;
  19     24     1  2            if nickels<>0 then
  20     25     1  2                BEGIN
  20     26     1  3                release:=release+2;
  21     27     1  3                nickels:=nickels-1
                                  END;
  23     29     1  2            if pennies<>0 then
  24     30     1  2                BEGIN
  24     31     1  3                release:=release+1;
  25     32     1  3                pennies:=pennies-1
                                  END;
  27     34     1  2          numberofcoins:=quarters+dimes+nickels+pennies;
  28     35     1  2          OUTWRD(130,release);
  29     36     1  2          END;
  31     37     1  1        END;     (*payment*)

  32     39     0  0      BEGIN   (*main*)
  32     40     0  1      INWRD(2,paid);
  33     41     0  1      INWRD(70,purchase);
  34     42     0  1      change  :=paid-purchase;
  35     43     0  1      coins   :=change mod 100;
  36     44     0  1      quarters:=coins div 25;
  37     45     0  1      coins   :=coins mod 25;
  38     46     0  1      dimes   :=coins div 10;
  39     47     0  1      coins   :=coins mod 10;
  40     48     0  1      nickels :=coins div 5;
  41     49     0  1      pennies :=coins mod 5;
  42     50     0  1      payment;
  43     51     0  1      END.     (*main*)
```

210469-5

Figure 2. Listing of CMAKER.86

```
(1) *BASE
    DECIMAL
(2) *MAP 0K LENGTH 32K HS
    *MAPIO 0T LENGTH 192T ICE
    *MAP
    MAP          0K LENGTH      32K HS
    MAP         32K LENGTH     992K GUARDED
    *MAPIO
    MAPIO 00000H LENGTH        000C0H ICE
    MAPIO 000C0H LENGTH        0FF40H USER
(3) *LOAD :F1:CMAKER.86
(4) *DEFINE POINTER begin = $
    *DEFINE BRKREG pay = :cmaker #9
    *DEFINE PROC display = DO
    .*WRITE USING (' "quarters = ",T,0,>')quarters
    .*WRITE USING (' "dimes = ",T,0')dimes
    .*WRITE USING (' "nickels = ",T,0,>')nickels
    .*WRITE USING (' "pennies = ",T,0')pennies
    .*RETURN TRUE
    .*END
(5) *GO USING pay
    ?UNIT 0 PORT 2H REQUESTS WORD INPUT (ENTER VALUE)*100
    ?UNIT 0 PORT 46H REQUESTS WORD INPUT (ENTER VALUE)*65
    *Probe 0 stopped at :CMAKER#9 + 4 because of execute break
      Break register is PAY Trace Buffer Overflow
(6) *quarters;dimes;numberofcoins
    +1
    +1
    +2
(7) *DEFINE SYSREG wr__number = WRITE AT .:cmaker.payment.numberofcoins &
    **CALL display
    *GO USING wr__number
      quarters = +1      dimes = +1
      nickels = +0       pennies = +0
    Probe 0 stopped at :CMAKER #28 + 3 because of bus break
      Break register is WR__NUMBER
(8) *numberofcoins
    +0
    *EVAL release
    1100Y 12T CH'..'
(9) *CLIPSOUT = 11Y
(10) *GO FOREVER
    ?UNIT 0 PORT 82H OUTPUT WORD 0C
    ?Probe 0 stopped at location 0033:00AEH because of bus not active
      Bus address = 0203DE
    *$=begin
    *
```

Figure 3. Sample Debugging Session (Explanations in Figure 4)

1. Checking to see that the default radix is decimal.
2. Mapping user program memory to I²ICE high-speed memory and user I/O ports to the I²ICE system console.
3. Loading the user program.
4. Defining debugging objects.

   The debugging variable **begin** is set to $, an I²ICE pseudo-variable representing the current execution point. At this point in the debugging session, $ is the beginning of the user program.

   The break register **pay** specifies a breakpoint at statement 9 in the user program.

   The debuggning procedure **display** displays the value of some user program variables on the console.
5. Beginning emulation with the debugging register **pay.** The console requests the two input values, **paid** and **purchase.** Then, the break occurs.
6. Displaying three user program variables.
7. Defining another debugging register. The specified event is the writing of the user program variable **numberofcoins.** When that event occurs, the I²ICE system calls the debugging procedure **display.** In addition to displaying some user program variables, this debugging procedure returns a Boolean value. Because this value is TRUE, the break occurs; if the value were FALSE, emulation would continue.
8. Displaying the two user program variables, **numberofcoins** and **release.** The EVAL command displays **release** in binary, decimal, hexadecimal, and ASCII. Unprintable ASCII characters appear as periods (.).
9. Asserting both output lines on the emulation clips. These lines are input to the prototype hardware and control a change release mechanism.
10. Resuming emulation. The console displays the write of **release** to the output port. The user program finishes executing, and the probe stops emulating because of bus inactivity. The $ is set back to the beginning of the user program in preparation for another emulation.

Figure 4. Explanation of Sample Debugging Session in Figure 3

I²ICE™ SYSTEM INSTRUMENTATION SUPPORT

I²ICE™ System Emulation Clips

Eight external input lines are sampled during each processor bus cycle. The I²ICE system records the values of these lines in the trace buffer during each execution cycle. The I²ICE system can use these values when defining events.

Four additional output lines synchronize I²ICE system events with external hardware. Two lines are active and programmable with I²ICE system commands. Two other lines, break and trace, allow an I²ICE system chassis to be linked to other I²ICE system chassis.

iPAT™ PERFORMANCE ANALYSIS TOOL

The Intel Performance Analysis Tool (iPAT analyst) helps software engineers optimize code and improve software reliability. Software object code generated by Intel assemblers and Intel compilers (e.g., for C, PL/M, Pascal, and FORTRAN) can be analyzed symbolically to improve software execution efficiency and to validate test coverage. Any object code that lacks Intel compiler information—but that can be run by Intel emulators and for which an absolute program map is available—can also be analyzed (non-symbolically) by the iPAT analyst. The iPAT analyst operation is currently supported via a target interface to the I²ICE system. For more information, see the iPAT analyst data sheet, Order Number 280165.

I²ICE™ SYSTEM SPECIFICATIONS

Host Requirements

Series III, Series IV, Model 800, or IBM PC AT or PC XT.

512K bytes in host development system memory space.

Two double-density diskette drives or a hard disk.

I²ICE™ System Software

I²ICE system host software
I²ICE system probe software
I²ICE system confidence tests
PSCOPE-86 (available as an option for the IBM PC AT or PC XT)

System Performance

Mappable zero wait-state memory (zero wait-states up to 10 MHz for 8086; 8 MHz for 8088; up to 10 MHz for 80186/80188, and up to 10 MHz for 80286):	Minimum 32K bytes Maximum 288K bytes
Trace buffer:	1023 x 48 bits
Virtual symbol table:	The number of user program symbols is limited only by available disk space.

Physical Characteristics

INSTRUMENTATION CHASSIS

Width: 17.0 in (43.2 cm)
Height: 8.25 in (21.0 cm)
Depth: 24.13 in (61.3 cm)
Weight: 48 lbs (21.9 kg)

HOST/CHASSIS CABLE

10 ft (3.0m) and 42 ft (12.8m) options for Series III/Series IV host
15 ft (4.6m) for PC host

INTER-CHASSIS CABLE SET

2 ft (61 cm) and 10 ft (3.0m) options

BUFFER BOX

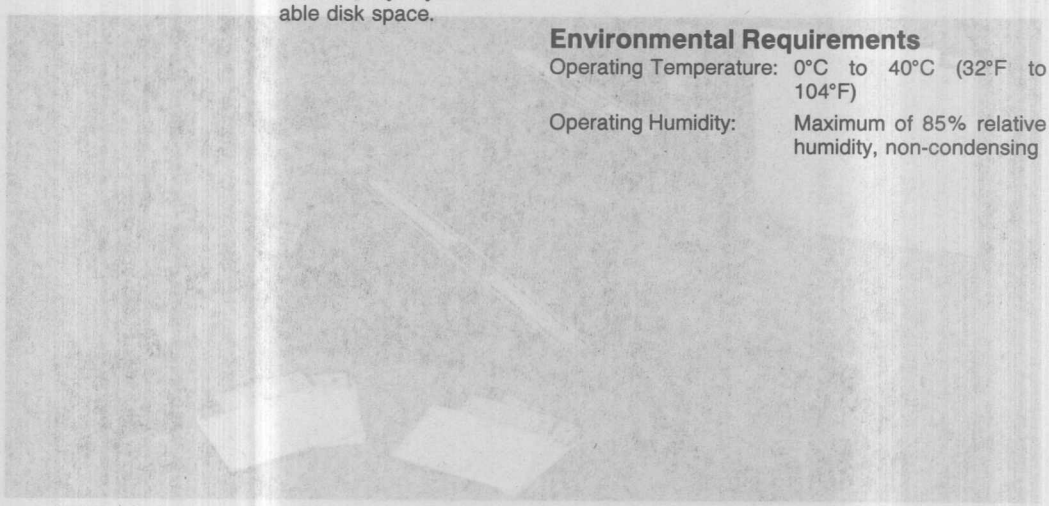
Width: 8.5 in (21.6 cm)
Height: 3.0 in (7.6 cm)
Depth: 10.0 in (25.4 cm)
Weight: 8 lbs (3.7 kg)

Electrical Characteristics

90–132V or 180–264V (selectable)
47–63 Hz
12 amps (AC)

Environmental Requirements

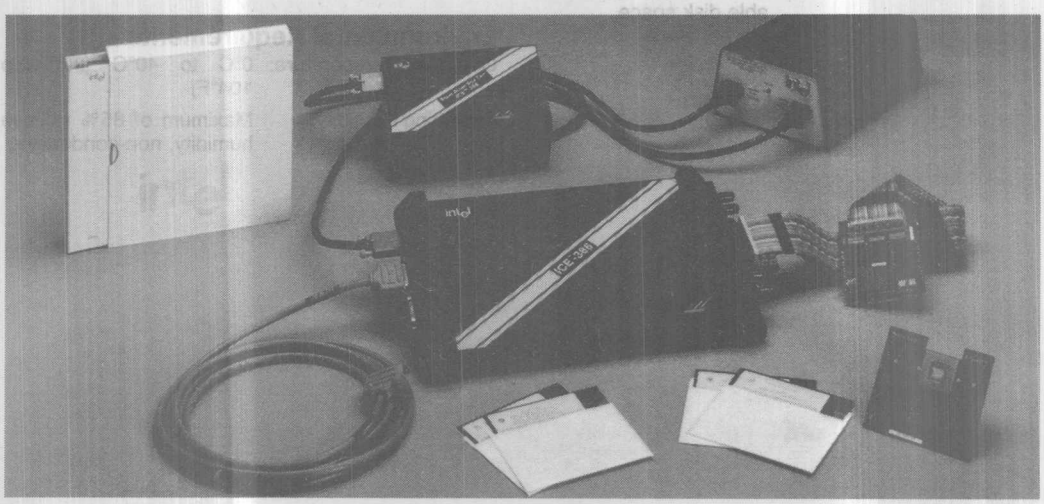
Operating Temperature:	0°C to 40°C (32°F to 104°F)
Operating Humidity:	Maximum of 85% relative humidity, non-condensing



ICE™-386
In-Circuit Emulator for the 80386

- **Provides Real-time Emulation of the 80386 at Speeds up to 16 MHz**
- **Maps User Program Memory into a Maximum of 128 KB of Memory in 4 KB Increments**
- **Allows Zero Wait-state Operation From User System Memory**
- **Provides Full Debug Support of 8086 Absolute Intel OMFs, and the 80286 and 80386 Bootloadable Intel OMFs**
- **Hosted on the Intel 286/310 System Running XENIX* or on the IBM PC AT Running DOS**
- **Allows User to Examine and Modify Memory, I/O, and the 80386 Registers**
- **Allows User to Examine and Modify Descriptor Tables, Page Tables, the Interrupt Table, and the Task State Segment**
- **Allows User to Set Breakpoints on Task Switching, External Inputs, Trace Buffer Full, and Instruction Execution**
- **Provides Real-time Execution Trace With Time Tags That Can Be Used to Analyze Execution Time of User Code**
- **Supports Coprocessor Debugging**
- **Stores over 2000 Frames of Program Execution History**
- **Allows User to Assemble and Disassemble Program Code in 80386 Instruction Mnemonics**
- **Provides Human Interface with Command Line Syntax, Guidance, Command Line Editing, Control Constructs, Debug Procedures, Debug Variables, and Shell Escape to the Host Operating System**
- **Uses the Four On-chip Breakpoints to Recognize Instruction Execution Addresses or Data Access Addresses Which Can Be Used to Trigger Break, Trace, or Change the State of External Sync Lines**

The ICE-386 In-Circuit Emulator provides sophisticated hardware and software debugging capabilities for 80386-based designs. These capabilities include emulation of the 80386 CPU and the ability to examine and modify registers in optional numeric coprocessors.



280316–1

*XENIX is a registered trademark of Microsoft Corporation.
‡UNIX is a registered trademark of AT&T Bell Laboratories.

October 1987
Order Number: 280316-002

PHYSICAL DESCRIPTION

The ICE-386 emulator consists of the following (see Table 1):

- Power supply with three power cables (an AC cable and two DC cables)
- Control unit (CU) with an RS-232C communication cable
- User cable assembly with Processor Module (PM)
- Stand-alone/self-test (SAST) unit
- Optional isolation board (OIB)
- A signal access board (SAB)
- SAB removal tool
- Software (includes the emulator software and diagnostic software)

The power supply powers the control unit and the stand-alone self-test unit (SAST).

The control unit (CU) controls the processor module and communicates with the host. The control unit has 128 KB of emulation memory, over 2000 frames of trace storage, and a control processor. The following connectors are on the control unit:

- Power supply connector
- GPIB connector
- RS-232C serial connector
- User cable connector
- Four BNC connectors, two for SYNCIN and two for SYNCOUT signals (used as qualifiers for triggering external events with the GO command).

The RS-232C cable or the (optional) GPIB cable connects the control unit to the host computer.

The user cable assembly comprises the processor module (the PM, which includes the 80386 emulation processor and IC buffers) and a cable that connects to the control unit. A target adaptor, mounted to the bottom of the PM, provides pin compatibility with the target 386 socket. The PM also has a socket for connecting the OIB or SAST unit to the emulator.

The stand-alone self-test (SAST) unit permits stand-alone operation of the ICE-386 emulator by supplying power and a 16MHz CLK2 to the processor module. The SAST unit provides self-test and diagnostic tests for the ICE-386 emulator and the optional isolation board (OIB). Sockets are provided for plugging the OIB into the SAST unit and also for using the ICE-386 emulator with math coprocessors.

The optional isolation board (OIB) is a circuit board that is installed between the processor module and the target system to protect the 80386 emulation processor from an untested target system. This ensures proper operation of the ICE-386 emulator even if the user bus fails. When it is determined that the target system will not electrically damage the 80386 emulation processor, the OIB can be removed and the emulation processor can operate at full speed. Use of the OIB limits the operating speed of the emulator to a maximum frequency of 8 MHz. See Table 3 for complete details on OIB timing.

The signal access board (SAB) provides access to all the 80386 signal pins for attaching external instruments, e.g., a logic analyzer. It can be installed between the processor module and the OIB board, or between the processor module and the target system.

The SAB removal tool is used to separate the signal access board from the processor module.

FUNCTIONAL DESCRIPTION

16 MHz Emulation

The ICE-386 emulates the 80386 at speeds up to 16 MHz, thus providing early detection of subtle timing problems that may arise at full speed. Intel's bond-out technology stresses the tightest possible conformance between timing parameters of the emulator and the target processor.

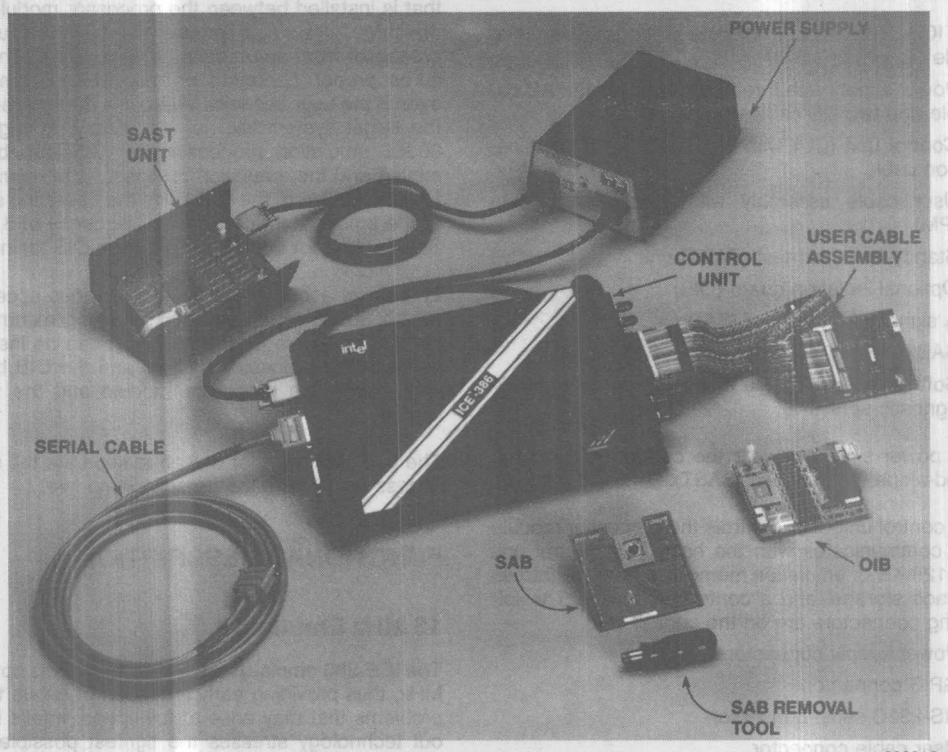
Event Recognition and Emulation Control

The ICE-386 emulator is capable of recognizing the following conditions:

- An instruction boundary
- A data write to a user-specified linear or virtual address
- A data access at a user-specified linear or virtual address
- Trace buffer full
- Task switch
- External input

Data breaks can be of BYTE, WORD, and DWORD length and are aligned on boundaries equal to the range size.

Table 1. ICE™-386 System Overview



280316-2

Name	Description
Control Unit	Contains an 80188 control processor, serial and GPIB communications, 128 KB mappable memory, and a trace buffer.
User Cable Assembly	Includes the user cable and the processor module, which contains the 80386 emulation processor and the OIB/SAST connector.
SAST Unit	Allows stand-alone operation and self-diagnosis of the ICE-386 emulator.
OIB	Protects the processor module from user-target-board bus failures during early prototype development.
SAB	Provides access to the labeled 80386 pins for use with external instruments.
SAB Removal Tool	Allows easy removal of the SAB from the emulation processor.

Based on the preceding events, the ICE-386 emulator can take the following actions:

- Halt emulation
- Sequence through events; maintain a count of such occurrences
- Assert SYNCOUT lines
- Turn trace on/off

In addition, software breakpoints, which allow breaks on instruction execution only, are useful in applications requiring extensive emulation breaking. These breakpoints have the following characteristics:

- Software breakpoints are placed only on RAM-based program code.
- Software breakpoints are valid in any addressing mode; however, if they are placed on a virtual/linear address, they will be translated to a physical address.
- Software breakpoints cannot be altered by the GO command; they can only be removed.

Memory Mapping

Memory can be mapped either to the user hardware (USER) or to the ICE-386 emulator (ICE). The 80386 microprocessor can address four gigabytes of memory space. Partitions anywhere within this four gigabytes processor space can be mapped to ICE. Thus, you can replace blocks of user address space to ICE and begin software development prior to completion of the target hardware. A maximum of 128 KB of mapped memory with an average of six wait states is mappable in 4 KB increments on 4 KB boundaries. Program code can be executed at zero wait-state from the target memory. When using mapped memory, paging cannot be enabled because both functions share the same resources on the 80386.

Mapping memory to ICE enables software development to begin before a user memory system is available. If the target system uses EPROMs, ICE-386 memory can replace the EPROM memory space so that you need not program the EPROMs during development.

Program Tracing

Over two thousand frames of program execution history can be stored in the trace buffer. Each trace frame holds a linear address of the branch address (or old TSS and new TSS for task switch), the byte count between branches, and a time-tag. This information is used to reconstruct a history of program execution. The ICE-386 emulator runs at about 93% of its full speed (when trace is enabled) assuming that a program discontinuity occurs every seven instructions. Turning the trace option off enables true full-speed emulation of the 80386 microprocessor. Trace data can also be displayed during emulation. However, enabling this feature may degrade emulator performance by causing the 80386 to enter a ready hang while the ICE-386 control processor is accessing the trace buffer.

Processor/Memory Examination and Modification

80386 registers can be accessed mnemonically (e.g., EAX) with the ICE-386 emulator software. Data can be displayed or modified in one of four bases: hexadecimal, decimal, octal, or binary, and in ASCII format. Program code can be disassembled and displayed as 80386 assembly instruction mnemonics, or it can be disassembled and displayed in 8086 or 80286 mode.

The 80386 microprocessor can operate in four modes: real, protected, page protected, and virtual 86. These modes modify the operation of the emulation processor to provide compatibility with the 8086, 80186, and 80286 microprocessors and to provide support for a variety of system architectures. The ICE-386 emulator can be used to debug in all of these microprocessor operating modes.

Program Stepping

With the ICE-386 emulator, you can single step through program code by referencing machine-level instructions, line numbers, or high-level language statements.

Symbolic Debugging

ICE-386 software takes advantage of the special debug information provided by Intel compilers, providing superior debug ability. It allows the software developer to examine or modify memory locations using symbolic references. A symbolic reference is a program procedure name, line number, or program label that corresponds to a location in the program space. Symbolic debugging allows you to work in the context of the original program, helping to meet the most critical schedules.

The ICE-386 emulator maintains a virtual table for program symbols making it possible for the table to exist without fitting entirely into the host RAM memory.

ICE™-386 Emulator Operating Modes

The ICE-386 emulator software has two operating modes, interrogation and emulation. Interrogation mode allows you to enter any ICE-386 command. Emulation mode allows execution of the user program code and execution of commands not requiring interaction with the emulation processor.

Debug Procedures

Debug procedures (PROCs) are named, user-defined groups of ICE-386 emulator commands. They can be stored on disk and recalled in later debugging sessions, thus saving you from having to re-enter commands.

One advantage of PROCs is that they allow you to automate the software testing process. The PROC may repeatedly generate test values, execute the user program with varied input values, and record the results. PROCs aid in the development of comprehensive batch tests.

ICE™-386 Human Interface Features

The ICE-386 emulator software includes features to help enter commands, set up the debug environment, and display command options. Also, there are features similar to those found in the UNIX‡/XENIX operating system environment. These features include input/output redirection, command piping, and escaping from the shell. The ICE-386 emulation software has a set of mathematical operators and control constructs similar to those found in Intel's I²ICE™ emulator and the C programming language.

Coprocessor Support

ICE-386 emulator commands provide access to the coprocessor's stack, status registers, and flags. In addition, the disassembly command extends to the math coprocessor's instructions and data types.

Language Support

The ICE-386 supports all 8086 absolute and 80286/80386 bootloadable Intel OMFs. These include the following Intel languages:

ASM86	ASM286	ASM386
PL/M-86	PL/M-286	PL/M-386
C-86	C-286	C-386
Pascal-86	Pascal-286	
FORTRAN-86	FORTRAN-286	

ICE-386™ EMULATOR COMMAND FUNCTIONS

The ICE-386 emulator command language can be divided into the following functional categories.

- Emulation commands — The GO, LSTEP, PSTEP, and ISTEP commands instruct the emulator to begin emulation. The GO command also has many options that allow recognition of a complex set of conditions.
- Utility commands — These are general purpose commands for use in a debugging environment. For example, the PRINT command displays selected frames of trace data. A command line editor and history buffer are also provided for ease of command entry.
- Environment commands — These commands set up the debug environment. For example, the MAP command sets up the memory map. The DEFINE command is used to create PROCs and definitions.
- File handling commands — These commands are used to access disk files. Debug object definitions can be saved to a disk file for use in later debug sessions. Debug sessions can be recorded to a disk file or a line printer for later analysis.
- Register access commands — These commands enable you to access the 80386 processor registers. Registers may be displayed or modified by name (e.g., EAX, EBX, etc.) or may be displayed in groups (REGS, CREGS, and SREGS).
- Descriptor access commands — These commands enable you to display descriptor tables and to symbolically display or modify individual descriptor components (e.g., GDT[10].limit, DT[CS].base).
- Memory and I/O access commands — These commands provide access to user memory space. Input and output are interpreted according to a set of data types and the requested numeric base.
- Stack frame access command — CALLSTACK allows you to display the current chain of procedure calls in the user program being executed.

SPECIFICATIONS

Host Specifications

The following lists the minimum host requirements for the Intel 286/310 and the IBM PC AT.

- Intel System 286/310 — 2 MB of RAM, 40 MB fixed disk, at least one floppy disk drive, and the XENIX 286 operating system (version 3.0 with Update 3).
- IBM PC AT — 2 MB of RAM (Lotus*, Intel, and Microsoft† extended memory specifications) (Intel's ABOVE Board with 1.5 MB is required), 20 MB fixed disk, at least one floppy disk drive, a serial interface, and the DOS operating system (version 3.1).

For enhanced upload/download performance, an optional IEEE-488 (GPIB) adaptor can be purchased for the IBM PC AT. Note that this IEEE-488 (GPIB) adaptor is reference sold by Intel and the details are given under the section titled "Referenced Product" after the section, "Ordering Information".

USER-INTERFACE SPECIFICATIONS

Figure 1 illustrates the interface between the 80386 processor in the processor module, the ICE-386, and the user system. When the processor module is directly attached to the target system without using the optional isolation board, the user must take three things into consideration: the shared data bus, the shared address bus, and the fact that the user-reset is intercepted by the ICE hardware. For ICE-386 to be functional in this configuration, the target system must satisfy the following restrictions:

- The user bus controller must only drive the data bus during a valid read cycle of the 80386 or while the processor is in hold state.
- The user system must gain control by asserting HOLD and receiving HOLDA before driving the address bus.
- The user reset signal is disabled during interrogation and delayed by 2 to 4 CLK2 cycles when in emulation.
- The user system must be capable of driving one additional TTL load on all signals that go to the ICE-386.

If the target system does not satisfy restrictions 1 and 2, the optional isolation board (OIB) must be used to isolate the emulation processor from the target system.

Required System Resources

The ICE-386 emulator requires exclusive use of the 80386's on-chip debug registers and INT1 under normal usage.

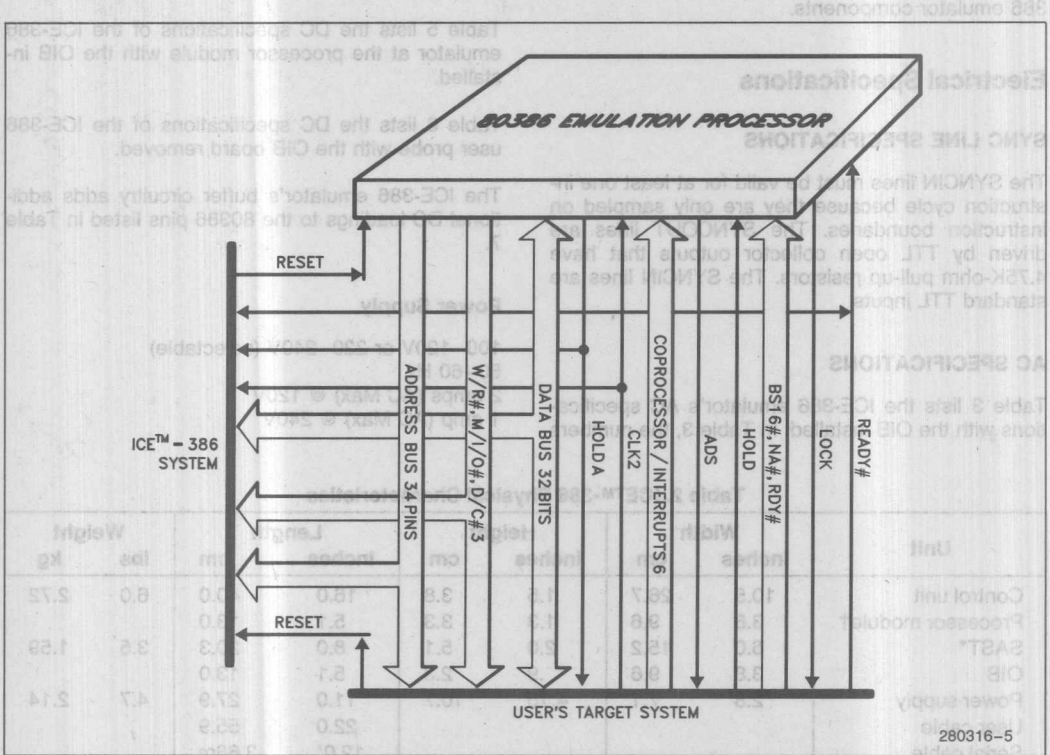


Figure 1. 80386-to-User System Interface

Mechanical Specifications

PROCESSOR MODULE/OPTIONAL ISOLATION BOARD DIMENSIONS

The height of the processor module (with OIB or SAB installed) requires 1¼ inches (3.18 cm) of space between boards to connect the processor module to the OIB, or to connect the processor module to the SAB. If the processor module is connected to both the OIB and the SAB, 1¼ inches (3.18 cm) are required on both sides, thus increasing the space requirement to 2½ inches (6.35 cm). Figure 2 shows the dimensions of the procesor module, the OIB and the SAB in the three possible configurations.

Table 2 lists the physical characteristics of the ICE-386 emulator components.

Electrical Specifications

SYNC LINE SPECIFICATIONS

The SYNCIN lines must be valid for at least one instruction cycle because they are only sampled on instruction boundaries. The SYNCOUT lines are driven by TTL open collector outputs that have 4.75K-ohm pull-up resistors. The SYNCIN lines are standard TTL inputs.

AC SPECIFICATIONS

Table 3 lists the ICE-386 emulator's AC specifications with the OIB installed. In Table 3, the numbers shown in the column titled "Maximum" are derived from using the timing symbols (t1, t2a, t2b, etc.) in the AC Specification Tables in the *80386 High Performance 32-Bit Microprocessor With Integrated Memory Management* data sheet and adding the additional time required to operate when the OIB is installed. The asterisks following the timing symbols in the column titled "Symbol" indicate that these timing symbols relate to the timing symbols of the 80386 microprocessor operating at 16 MHz. Active low is indicated by the overscore (e.g., $\overline{\text{LOCK}}$).

The AC specifications for the ICE-386 emulator with the OIB removed are the same as those for the 80386 microprocessor, with the 01B removed, except for t25 and t26, as shown in Table 4.

DC SPECIFICATIONS

Table 5 lists the DC specifications of the ICE-386 emulator at the processor module with the OIB installed.

Table 6 lists the DC specifications of the ICE-386 user probe with the OIB board removed.

The ICE-386 emulator's buffer circuitry adds additional DC loadings to the 80386 pins listed in Table 7.

Power Supply

100–120V or 220–240V (selectable)
50–60 Hz
2 amps (AC Max) @ 120V
1 amp (AC Max) @ 240V

Table 2. ICE™-386 Physical Characteristics

Unit	Width		Height		Length		Weight	
	inches	cm	inches	cm	inches	cm	lbs	kg
Control unit	10.5	26.7	1.5	3.8	16.0	40.0	6.0	2.72
Processor module†	3.8	9.6	1.3	3.3	5.1	13.0		
SAST*	6.0	15.2	2.0	5.1	8.0	20.3	3.5	1.59
OIB	3.8	9.6	.9	2.3	5.1	13.0		
Power supply	2.8	7.1	4.15	10.7	11.0	27.9	4.7	2.14
User cable					22.0	55.9		
Serial cable					12.0′	3.66m		
SAB	4.6	11.7	.8	2.0	4.1	10.7		

†Measurement includes target adapter
*Measurement includes user cable

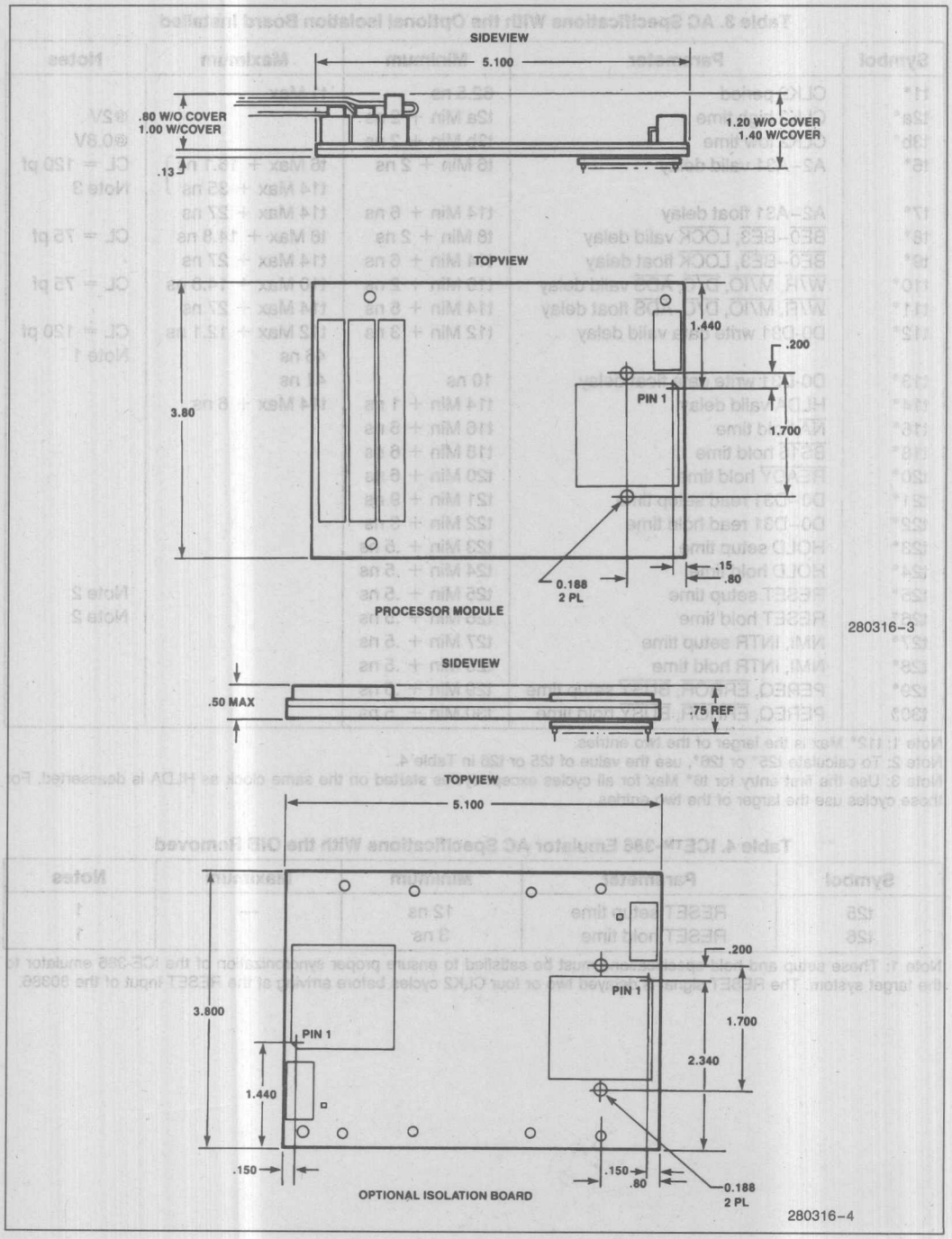


Figure 2. Board Dimensions

Table 3. AC Specifications With the Optional Isolation Board Installed

Symbol	Parameter	Minimum	Maximum	Notes
t1*	CLK2 period	62.5 ns	t1 Max	
t2a*	CLK2 high time	t2a Min + 2 ns		@2V
t3b*	CLK2 low time	t3b Min + 2 ns		@0.8V
t6*	A2–A31 valid delay	t6 Min + 2 ns	t6 Max + 16.1 ns t14 Max + 35 ns	CL = 120 pf Note 3
t7*	A2–A31 float delay	t14 Min + 6 ns	t14 Max + 27 ns	
t8*	$\overline{BE0}$–$\overline{BE3}$, $\overline{LOCK}$ valid delay	t8 Min + 2 ns	t8 Max + 14.8 ns	CL = 75 pf
t9*	$\overline{BE0}$–$\overline{BE3}$, $\overline{LOCK}$ float delay	t14 Min + 6 ns	t14 Max + 27 ns	
t10*	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ valid delay	t10 Min + 2 ns	t10 Max + 14.8 ns	CL = 75 pf
t11*	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ float delay	t14 Min + 6 ns	t14 Max + 27 ns	
t12*	D0-D31 write data valid delay	t12 Min + 3 ns	t12 Max + 12.1 ns 46 ns	CL = 120 pf Note 1
t13*	D0-D31 write data float delay	10 ns	41 ns	
t14*	HLDA valid delay	t14 Min + 1 ns	t14 Max + 6 ns	
t16*	$\overline{NA}$ hold time	t16 Min + 6 ns		
t18*	$\overline{BS16}$ hold time	t18 Min + 6 ns		
t20*	$\overline{READY}$ hold time	t20 Min + 6 ns		
t21*	D0–D31 read setup time	t21 Min + 9 ns		
t22*	D0–D31 read hold time	t22 Min + 3 ns		
t23*	HOLD setup time	t23 Min + .5 ns		
t24*	HOLD hold time	t24 Min + .5 ns		
t25*	RESET setup time	t25 Min + .5 ns		Note 2
t26*	RESET hold time	t26 Min + .5 ns		Note 2
t27*	NMI, INTR setup time	t27 Min + .5 ns		
t28*	NMI, INTR hold time	t28 Min + .5 ns		
t29*	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$ setup time	t29 Min + .5 ns		
t30*	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$ hold time	t30 Min + .5 ns		

Note 1: t12* Max is the larger of the two entries.
Note 2: To calculate t25* or t26*, use the value of t25 or t26 in Table 4.
Note 3: Use the first entry for t6* Max for all cycles except cycles started on the same clock as HLDA is deasserted. For those cycles use the larger of the two entries.

Table 4. ICE™-386 Emulator AC Specifications With the OIB Removed

Symbol	Parameter	Minimum	Maximum	Notes
t25	RESET setup time	12 ns	—	1
t26	RESET hold time	3 ns		1

Note 1: These setup and hold specifications must be satisfied to ensure proper syncronization of the ICE-386 emulator to the target system. The RESET signal is delayed two or four CLK2 cycles before arriving at the RESET input of the 80386.

Table 5. ICE™-386 Emulator DC Specifications With OIB Installed

Symbol	Parameter	Minimum	Maximum	Notes
OIB-I_{cc}	Supply current		PM-I_{cc} + 500 mA	1
V_{ol}	Output low voltage. I_{ol} = 48 ma, A2-A31, D0-D31, $\overline{BE0}$-$\overline{BE3}$, $\overline{ADS}$, $\overline{W/R}$, $\overline{D/C}$, $\overline{M/IO}$, $\overline{LOCK}$, HLDA		0.5V	
V_{oh}	Output high voltage. I_{oh} = 3ma, A2-A31, D0-D31, $\overline{BE0}$-$\overline{BE3}$, $\overline{ADS}$, $\overline{W/R}$, $\overline{D/C}$, $\overline{M/IO}$, $\overline{LOCK}$, HLDA	2.4V		
I_{il}	Input low current. CLK2, RESET, HOLD $\overline{READY}$		.5 mA .26 mA	
I_{ih}	Input high current. CLK2, RESET, HOLD $\overline{READY}$		20 μa 40 μa	
I_{lo}	Output leakage current. A2-A31, D0-D31, $\overline{BE0}$-$\overline{BE3}$, $\overline{ADS}$, $\overline{W/R}$, $\overline{D/C}$, $\overline{M/IO}$, $\overline{LOCK}$		± 20 μa	

Note 1: OIB-I_{cc} is the supply current of the ICE-386 user probe with the OIB installed. PM-I_{cc} is the supply current of the user probe with the OIB removed.

Table 6. ICE™-386 Emulator DC Specifications Without OIB Installed

Symbol	Parameter	Maximum	Notes
PM-I_{cc}	Supply current. CLK2 = 16 MHz: with 80386-16	80386-I_{cc} + 800 mA	
C_{in}*	Input capacitance	C_{in} + 15 pf	1, 2
C_{out}*	Output or I/O capacitance	C_{out} + 15 pf	1, 3
C_{clk}*	CLK2 Capacitance	C_{clk} + 15 pf	1

Note 1: Symbols followed by an asterisk are ICE-386 user-probe specifications and are given in terms of the corresponding 80386 specifications plus a constant.
Note 2: Specification applies to the $\overline{READY}$ and RESET signals.
Note 3: Specification applies to the A2-A31, D0-D31, $\overline{BE0}$–$\overline{BE3}$, $\overline{W/R}$, $\overline{M/IO}$, $\overline{ADS}$, and HLDA signals.

Table 7. Additional DC Loading

Signal	I_{ih} Maximum	I_{il}Maximum
A2-A31, D0-D31, $\overline{BE2}$, $\overline{BE3}$	0.02 mA	0.1 mA
$\overline{ADS}$, $\overline{BE0}$, $\overline{BE1}$, HLDA	0.02 mA	1 mA
$\overline{M/IO}$	0.02 mA	0.5 mA
$\overline{W/R}$	0.04 mA	1 mA
$\overline{READY}$	0.02 mA	0.02 mA
CLK2	0.05 mA	2 mA

Note: For more information on these signals, refer to the 80386 data sheet.

Environmental Specifications

Operating temperature — 10°-40°C (50°-104°F)

Operating humidity — maximum of 85% relative humidity, non-condensing

DOCUMENTATION

ICE™-386 In-Circuit Emulation User's Guide, Order Number 166182

80386 High Performance 32-Bit Microprocessor with Integrated Memory Management, data sheet, Order Number 231630

ORDERING INFORMATION

Order Code	Description
ICE386XP	The complete ICE-386 emulator system including control unit, processor module, power supply, SAST, OIB, SAB, a serial communication cable (SCOM5), and software (S386XP/S386X). (Requires software license, Class I)
ICE386DP	The complete ICE-386 emulator system including control unit, processor module, power supply, SAST, OIB, SAB, a serial communication cable (SCOM4), and software (S386DP/S386D). (Requires software license, Class I). Note: A GPIB adapter card and cable for the IBM PC AT is optionally available. Please contact your Intel Field Applications Engineer for more information.
SCOM4	A 12-foot serial cable for the ICE-386 — IBM PC AT connection
S386DP	ICE-386 software for hosting on the IBM PC AT running DOS3.1
SCOM5	A 12-foot serial cable for the ICE-386 — Intel 286/310 connection
S386XP	ICE-386 software for hosting on the Intel System 286/310 running XENIX release 3.0 with Update 3

Reference Product

Please call National Instruments at 800-531-5066 or 800-474-2488 (in Texas only) to place your order in U.S. contact your Intel Salesman for a distributor nearest you.

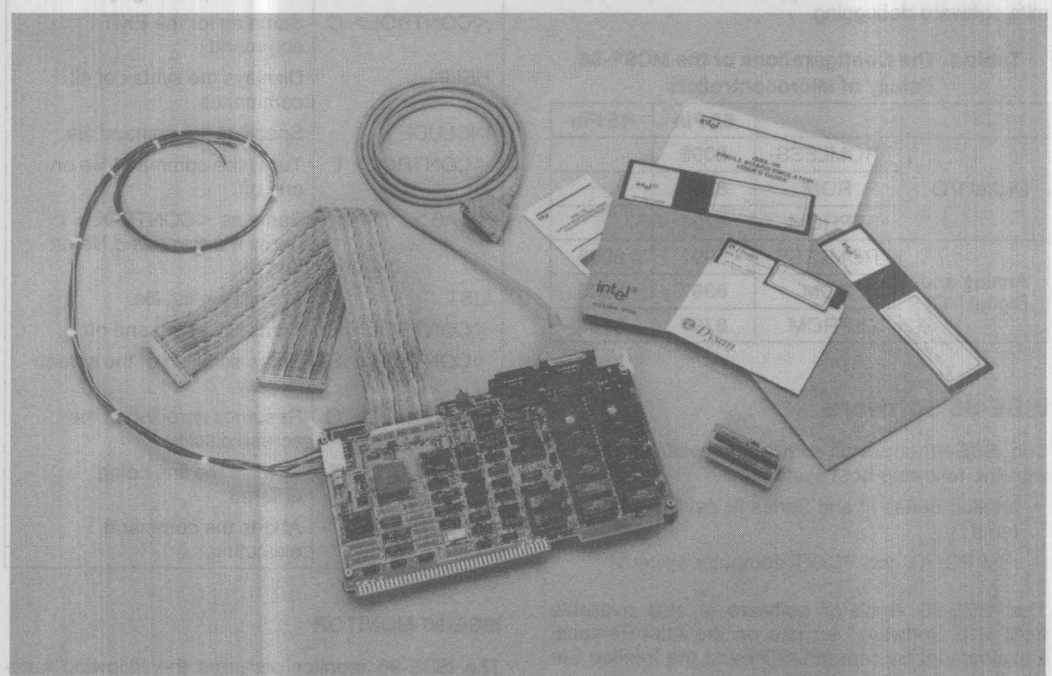
GPIB-PCIIA	Part No. 776113-01. GPIB adaptor card and DOS handler for the IBM PC AT.
Cable	Part No. 763061-01 for 1 meter cable
	Part No. 763061-02 for 2 meter cable
	Part No. 763061-03 for 4 meter cable

iSBE-96 DEVELOPMENT KIT
SINGLE BOARD EMULATOR AND ASSEMBLER FOR THE MCS®-96 FAMILY OF MICROCONTROLLERS

- **Hosts**
  - **— Intellec® Series III/IV Development Systems**
  - **— IBM* PC AT, PC XT, and Compatibles (3.0)**
- **Eight Software Execution Breakpoints That Can Selectively Be Turned On and Off**
- **12 MHz Emulation Speed**
- **Single Line Assembler/Disassembler**
- **MCS®-96 Software Support Package**
- **Configurable Serial I/O**
- **17.75 of On-Board User Memory**
- **Optionally Expandable to 64K of On-Board User Memory**

The iSBE-96 emulator supports the execution and debugging of programs for the MCS-96 family of microcontrollers at speeds up to 12 MHz. The MCS-96 family configurations are shown in Table 1. The iSBE-96 emulator consists of an 8097 microcontroller, a serial port and cables, and an EPROM-based monitor that controls emulator operation and the user interface.

The iSBE-96 emulator is a combination of hardware and software that permits programs written for the MCS-96 family of microcontrollers to be run and debugged in the emulator's artificial environment or in the user's prototype system. As a result, development time can be reduced by the early integration of hardware and software.



231015–1

April 1987
Order Number: 231015-006

FUNCTIONAL DESCRIPTION

Integrated Hardware and Software Development

The iSBE-96 emualtor allows hardware and software development to proceed simultaneously. This approach is more time- and cost-effective than the alternate method: independent hardware and software development followed by system integration. With the iSBE-96 emulator, prototype hardware can be added to the system as it is designed; software and hardware integration occurs while the product is being developed. The emulator aids in the recognition of hardware and software problems.

Emulation is the controlled execution of the prototype software in the prototype hardware or in an artificial hardware environment that duplicates the microcontroller of the prototype system. The iSBE-96 emulator permits reading and writing of system memory, and control of program execution. The emulator also allows interactive debugging of the prototype software and can externally control program execution while operating in the prototype system. When the prototype system memory is not yet available, the iSBE-96 emulator's on-board memory permits software debugging.

Table 1. The Configurations of the MCS®-96 Family of Microcontrollers

		68 Pin	48 Pin
Digital I/O	ROMLESS	8096	
	ROM	8396	
	EPROM	8796	
Analog and Digital I/O	ROMLESS	8097	8095
	ROM	8397	8395
	EPROM	8797	8795

iSBE-96 Software

The iSBE-96 emulator software is available for use with the following host systems:

- Intellec Series III and Series IV development systems
- IBM PC/AT and PC/XT computer systems

The iSBE-96 emulator software is also available from U S Software* for use on the Intel Personal Development System (iPDS™) and the Intellec Series II development system.

***NOTE:**
U S Software is a registered trademark of United States Software Corporation.

The iSBE-96 emulator is supplied with a driver routine that communicates with the monitor software on the iSBE-96 emulator board through serial channel 1 or 2 (com1/com2). The driver interrupts the 8097 using the non-maskable interrupt (NMI) line for incoming keyboard input. The commands associated with the driver and the monitor are described in the following sections.

iSBE-96 Driver

iSBE-96 emulator is shipped with driver software for use on the Series III/IV development systems and the IBM PC AT/XT running PC DOS, version 3.0 or greater. The driver software provides a few easy-to-use commands. These commands are described in Table 2. ASM/DASM available on DOS version only.

Table 2. iSBE-96 Driver Commands

Driver Command	Function
ASM	Loads memory with MCS-96 assembly mnemonics.
DASM	Displays memory as MCS-96 assembly mnemonics.
EXIT	Exits the driver and returns to the host operating system.
<CONTROL> C	Same as for the EXIT command.
HELP	Displays the syntax of all commands.
INCLUDE	Specifies a command file.
<CONTROL> I	Turns the command file on and off.
<TAB>	Same as <CONTROL> I (turns the command file on and off).
LIST	Specifies a list file.
<CONTROL> L	Turns list file on and off.
<CONTROL> S	Stops scrolling of the screen display.
<CONTROL> Q	Resumes scrolling of the screen display.
<CONTROL> X	Deletes the line being entered.
<ESCAPE>	Aborts the command executing.

iSBE-96 MONITOR

The iSBE-96 monitor performs the following functions:

- Loads and saves user programs.
- Independently emulates user programs.

Table 3. iSBD Monitor Commands

Monitor Command	Function
BAUD	Sets up the baud rate.
BR	Permits display and setting of up to eight software breakpoints.
BYTE	Permits display and changing of a single byte or range of bytes of memory or a single byte of the 8097 internal registers.
CHANGE	Permits display and changing of a series of memory words or bytes.
<CONTROL> S	Stops scrolling of the screen display.
<CONTROL> Q	Resumes scrolling of the screen display.
<CONTROL> X	Deletes the line being entered.
<ESCAPE>	Aborts the command executing.
GO	Begins emulation and continues until an enabled breakpoint is reached or the escape key is pressed.
LOAD	Loads programs and data from disks.
MAP	Permits mapping of several preprogrammed memory maps; also permits configurable serial I/O and selective servicing of the watchdog timer.
PC	Displays and changes the program counter.
PSW	Displays and changes the program status word.
RESET CHIP	Resets the 8096 to power-up conditions.
SAVE	Saves programs and data to disks.
SP	Displays and changes the stack pointer.
STEP	Provides single-step emulation with selective display formats.
VERSION	Displays the monitor version number.
WORD	Permits display and changing of a single word or range of words of memory or a single word of the 8097 internal registers.

- Examines and changes memory contents.
- Examines registers.
- Maps the file capabilities of the serial ports (DS/DT).
- Maps different memory configurations.

The monitor commands are described in Table 3.

Integrating Hardware and Software

When the prototype system hardware is developed, the iSBE-96 emulator interfaces to the prototype through two 50-pin ribbon cables. The emulator can then execute code from the iSBE-96 on-board RAM (or from user-provided memory) and exercise the prototype system hardware.

BLOCK DIAGRAM

Figure 1 is a block diagram showing the iSBE-96 emulator. The following sections describe each block.

The Processor

The 68-pin processor of the iSBE-96 emulator is used only in the 8097 external-access mode. An 8097BH will be supported in 16-bit bus mode only.

An adapter board is provided for the 68-pin PGA version of the 8096 and 8097 microcontrollers. When debugging a 68-pin package, the two 50-pin ribbon cables plug into the 68-pin adaptor board which is plugged into the 68-pin socket on the prototype system.

When debugging a 48-pin package, the two 50-pin cables plug into the 48-pin adaptor board, which is then plugged into a 48-pin socket in the prototype system. A 68-pin PLCC Adaptor may be ordered.

iSBE-96 Emulator I/O

The iSBE-96 emulator's memory-mapped I/O devices are used to manage the system. These I/O devices are mapped into memory between locations 01F00H and 01FFFH.

Included as part of the I/O are two serial ports. One is configured as data set (DS) and the other as data terminal (DT). When operating with an Intellec® development system, the data set port is used as the system console and the link for exchanging files.

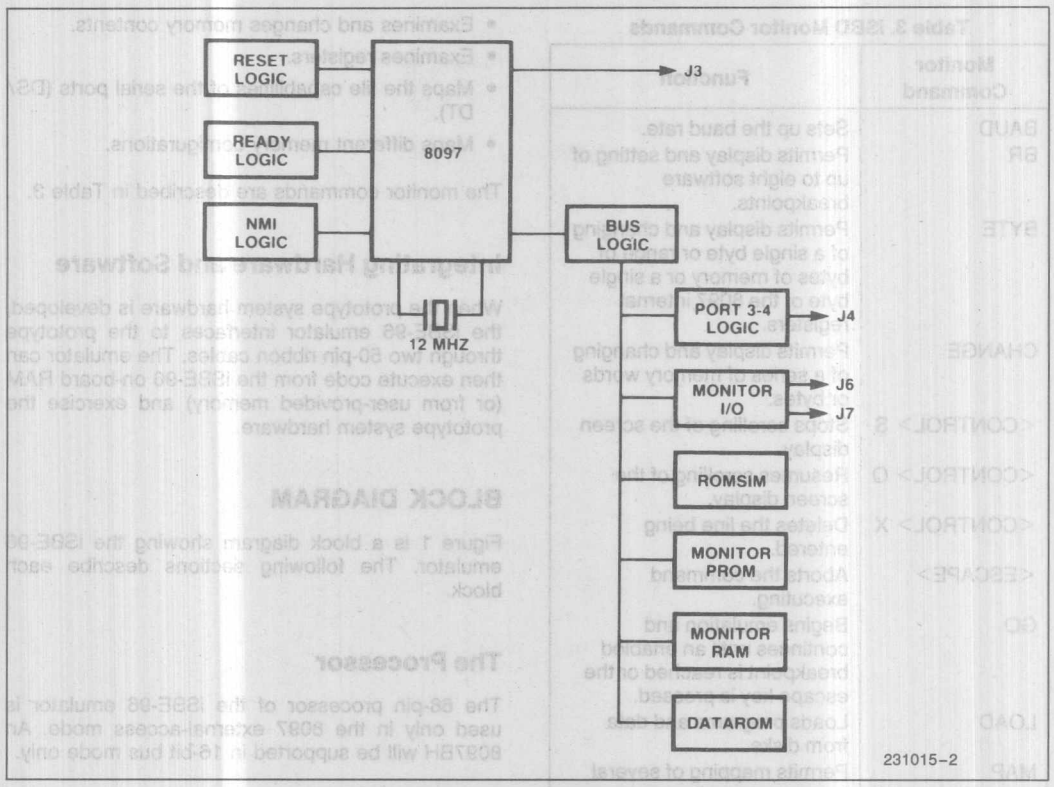


Figure 1. Block Diagram for the iSBE-96 Single Board Emulator

The serial ports are serviced under control of the NMI interrupt. The NMI interrupt has highest priority on the microcontroller and interrupts the user program when characters are entered from the keyboard. When in emulation, the monitor will still service inputs from the keyboard and execute certain monitor commands. Monitor activity is not transparent to the user.

Simulated ROM (ROMSIM)

There are eight 28-pin JEDEC byte-wide sockets with 2K-by-8 static RAMS present on the board. The partition on the user's prototype system that will be ROM is simulated by RAM on the iSBE-96 emulator board. This RAM facilitates easy program development, allowing users to correct and test problems in their programs.

ROMSIM can be expanded by replacing the iSBE-96 RAMs with 8K-by-8 static RAMs.

Port 3-4 Logic

The port 3-4 logic has two functions: to provide bus expansion and to provide I/O ports. The port 3-4 logic is controlled by a software switch available with the MAP command.

The iSBE-96 emulator reconstructs ports 3 and 4 of the 8395, 8396, and 8397 microcontrollers when the logic is defined by the MAP command as port 3-4. This port function should be selected when one of these four microcontrollers is intended as the target microcontroller.

When the BUS switch of the MAP command is specified, the iSBE-96 address/data expansion bus is available to the prototype system.

THE iSBE-96 EMULATOR MEMORY MAP

The target system should be designed with a memory map that is compatible with one of the iSBE-96

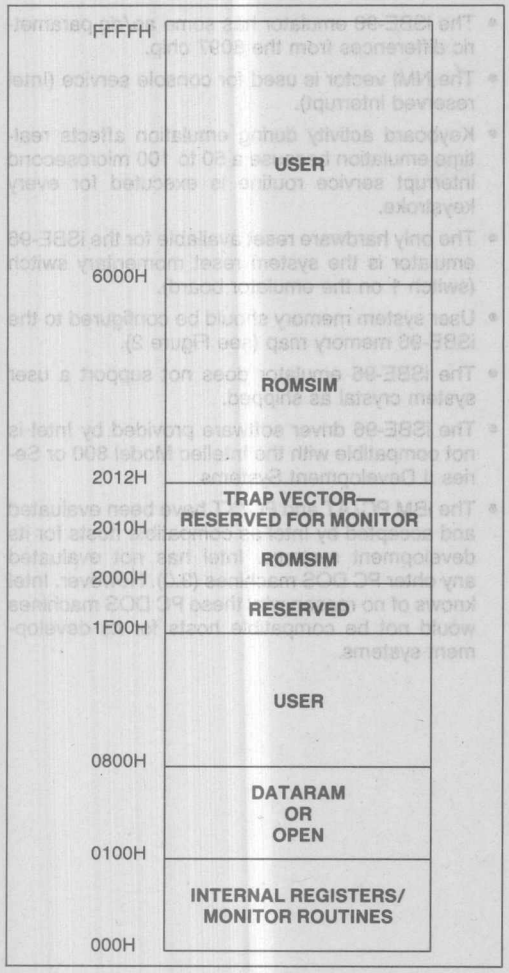


Figure 2. iSBE-96 Emulator Default Mapping

memory maps. Figure 2 shows the default address mapping. The following sections describe the areas of memory.

Internal Registers/Monitor Routines

Normally locations 000H through 0FFH contain the internal register space of the 8097. However, instruction fetches from these locations access external memory. This memory space contains the monitor's non-maskable interrupt service routine and utility routines.

For the monitor to access the user memory, the address and data is passed to the interrupt or utility routines. The routines then modify the mode register to enable user memory, disable all of the monitor's memory (except page zero), and perform the appropriate operation. After an operation is complete, the service and utility routines restore the mode register to its previous state and return to the main monitor code. The NMI service routine is used to handle the keyboard input on the serial port.

DATARAM

Locations 100H to 7FFH are mapped as the DATARAM space. This RAM is for general purpose use and is optionally enabled by using the MAP command. When the DATARAM buffer is not enabled, any access to this partition results in an access to user prototype system memory.

User Area

Locations 800H to 1EFFH are a user area. If an access is made to this partition, it is directed to the user's prototype system. Any memory mapped as I/O in the user system should be placed in this partition. With 8K-by-8 static RAMs, this area is located and available on the iSBE-96 board.

Reserved Area

Locations 1F00H to 1FFFH are reserved by the monitor for on-board I/O devices.

ROMSIM

Because some of the MCS-96 family of microcontrollers are ROMLESS parts, a user program can be loaded for execution into the on-board RAMS of the iSBE-96 emulator. Locations 2000H to 5FFFH are mapped to this RAM space; the space is called ROMSIM.

Trap Vector

Locations 2000H to 2010H are the interrupt vector locations. Vector address location 2010H is used by the iSBE-96 monitor for breakpoints.

User Area

The partition 6000H to 0FFFFH is mapped to the user prototype area. During emulation any access to this partition is directed to the user's prototype system.

EXPANDING ON-BOARD MEMORY

On-board memory can be expanded to a full 64K bytes by replacing the supplied 2K-by-8 static RAMs with 8K-by-8 static RAMs or PROMs. The user may also replace on-board ROMSIM memory with 2K-by-8 PROMs or even locate all 64K bytes of memory on the prototype system.

DESIGN CONSIDERATIONS

Designers should note the following considerations for designing with the iSBE-96 emulator:

- The iSBE-96 software uses 6 bytes of user stack space.
- Analog signal accuracy is impaired when driven over the emulator cable (up to ±50 mV loss of A/D conversion accuracy).
- The iSBE-96 emulator has some ac/dc parametric differences from the 8097 chip.
- The NMI vector is used for console service (Intel reserved interrupt).
- Keyboard activity during emulation affects real-time emulation because a 50 to 100 microsecond interrupt service routine is executed for every keystroke.
- The only hardware reset available for the iSBE-96 emulator is the system reset momentary switch (switch 1 on the emulator board).
- User system memory should be configured to the iSBE-96 memory map (see Figure 2).
- The iSBE-96 emulator does not support a user system crystal as shipped.
- The iSBE-96 driver software provided by Intel is not compatible with the Intellec Model 800 or Series II Development Systems.
- The IBM PC/AT and PC/XT have been evaluated and accepted by Intel as compatible hosts for its development systems. Intel has not evaluated any ohter PC DOS machines (3.0). However, Intel knows of no reason why these PC DOS machines would not be compatible hosts for its development systems.

SPECIFICATIONS

Equipment Supplied

Standard MULTIBUS®-size board assembly

EPROM-based monitor

Auxiliary power cable

RS-232 serial cables

Two standard, 18 in., 50-pin ribbon cables for connection to the user's prototype system

Adapter board for the 48-pin DIP and 68-pin PGA versions of the MCS-96 microcontroller

MCS-96 software support package

One 8 in. single-density software disk for the Series III

One 8 in. double-density software disk for the Series III

One 5¼ in. software disk for the Series IV

One 5¼ in. software disk for the IBM PC AT/XT

Documentation

iSBE-96 User's Guide (Order number 164116)

iSBE-96 Pocket Reference (Order number 164157)

Developing MCS-96 Applications Using iSBE-96 (Order Number 280249-001, AP-273)

Emulation Clock

12 MHz supplied crystal

Physcial Characteristics

Width: 6.75 in. (17.15 cm)
Length: 12 in. (30.48 cm)
Height: 0.75 in. (1.91 cm)

DC Electrical Requirements

Voltage	Current
+5V ± 5%	3.5a max
+12V ± 5%	0.06a max
−12V ± 5%	0.05a max

Environmental Characteristics

Operating Temperature: 10°C to 40°C

Operating Humidity: 10% to 85% relative humidity, without condensation

IBM PC XT/AT Host Requirements

- PC DOS, version 3.0 or greater
- External power supply
- Serial channel Com1/Com2

ORDERING INFORMATION

Intel 3065 Bowers Ave.
Santa Clara, CA 95051

Part Number	Description
SBE96SKIT	iSBE-96 single board emulator for use with the Series III/IV development systems. The kit contains the following parts: • iSBE-96 single board emulator • MCS-96 software support package for the Series III/IV development systems • iSBE-96 Series III/IV upgrade kit (cables and software needed to run on Intel Hosts)
SBE96DKIT	iSBE-96 single board emulator for use with the IBM PC/AT and PC/XT computer systems. The kit contains the following parts: • iSBE-96 single board emulator • MCS-96 software support package for PC DOS • iSBE-96 DOS upgrade kit (cables and software needed to run on the IBM PC/AT or PC/XT)
SBE96DU	iSBE-96 DOS upgrade kit for those customers who wish to upgrade their Series III/IV kit to run on the IBM PC AT or PC XT.
SBE96SU	iSBE-96 Series III/IV upgrade kit for those customers who wish to upgrade their DOS kit to run on Intel Hosts).
TASBEE	68-pin PLCC Adaptor Board.

U S Software
5470 N. W. Innisbrook
Portland, OR 97229
Phone: 503-645-5043
International Telex 4993875

Part Number	Description
XASM96	Performs assembly of MCS®-96 programs written on the iPDS.
ATOP96	iPDS and Series II software for iSBE-96 host communications. Performs host communications and assembly/disassembly of iSBE-96 instructions. The XASM Host Cross Assembler software must be ordered with this software.

VLSiCE™-96
IN-CIRCUIT EMULATOR FOR THE 8X9X FAMILY OF MICROCONTROLLERS

- **Precise Real-Time Emulation of the 8X9X Family of Components**
- **64K of Mappable Memory for Early Software Debug and (EP)ROM Simulation**
- **A 4K-Entry Trace Buffer for Storing Real-Time Execution History, Including Both Code and Data Flows**
- **FASTBREAKS and Dynamic Trace**
- **Symbolic Debugging Allows Accesses to Memory Locations and Program Variables (Including Dynamic Variables) Using Program-Defined Names from the User's Assembler or Compiler Source Code**
- **Shadow Registers Allow Reading Many 8096 Write-Only and Writing Many Read-Only Registers**
- **Break and Trace are Qualified on Execution Addresses, Data Addresses, and Values (Both External and Internal RAM), Opcodes, Selected PSW Flags, and 2 External Sync Lines**
- **Equipped with the Integrated Command Directory (ICD™) Which Provides**
  - **— An On-Line Help File**
  - **— A Dynamic Syntax Menu**
  - **— Dynamic Command-Entry**
  - **— Error Checking**
  - **— On-Line Editor**
- **Serially Linked to Intel Series III/IV Hosts or IBM* PC-XT and AT**

The VLSiCE-96 In-Circuit Emulator is a debugging and test tool used for development of the hardware and software of a target system based on the 8X9X family of microcontrollers (8095, 8096, 8097, 8395, 8396, 8397, 8795, 8796, 8797) including BH components.



280140–1

*IBM is a trademark of International Business Machines.

October 1987
Order Number: 280140-004

INTRODUCTION

The VLSiCE-96 emulator allows hardware and software development of a design project to proceed simultaneously. With the VLSiCE-96 emulator, prototype hardware can be added to the system as it is designed and software can be developed prior to the completion of the hardware prototype. Thus, software and hardware can be integrated while the product is being developed.

The VLSiCE-96 emulator assists four stages of development:

- Software development
- Hardware development
- System integration
- System test

Software Development

The VLSiCE-96 emulator can be operated without being connected to a prototype or before any of the prototype hardware is available. In this stand-alone mode, the VLSiCE-96 emulator can be used to facilitate application program development.

Hardware Development

Because the VLSiCE-96 emulator precisely matches the component's electrical and timing characteristics as well as full bus access, it is a valuable tool for hardware development and debug.

System Integration

Integration of software and hardware begins when the microcontroller socket is connected to any functional element of the target system. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software can be system tested with the VLSiCE-96 emulator in real-time operation as it becomes available.

System Test

When the prototype is complete, it is tested with the final version of the system software. The VLSiCE-96 emulator can then be used to verify or debug the target system as a completed unit.

By providing support for the ROMLESS, ROM, and EPROM versions of the microcontroller, the VLSiCE-96 emulator has the ability to debug a prototype or production product at any stage in its development without introducing extraneous hardware or software test tools.

PHYSICAL DESCRIPTION

The VLSiCE-96 emulator consists of the following components (see Figure 1):

- Software (includes the VLSiCE-96 emulator software, diagnostic software, and tutorial)
- 68-pin PGA adaptor
  68-pin PLCC adaptor (optional)
  48-pin DIP adaptor (optional)
- Controller pod
- User cable assembly (consisting of the user cable and processor module)
- Serial cable (host-specific)
- Crystal power accessory (CPA)
- Multi-synchronous accessory (MSA) (optional)
- Power supply and V_{CC} booster module
- AC and DC power cables

VLSiCE-96 software fully supports all mnemonics, object file formats, and symbolic references generated by Intel's ASM-96, PL/M-96, and C-96.

The on-line tutorial is written in VLSiCE-96 command language. Thus, the user is able to interact with and use the VLSiCE-96 emulator while executing the tutorial.

The controller pod contains 64K of ICE memory, a 4K-entry trace buffer, and circuitry that provides communication between the host and the emulator.

The processor module contains a special version of the Intel 8096 microcontroller, called the emulation processor. This chip performs real-time and single-step execution of a program's object code for execution and debugging purposes in place of the target system microcontroller.

The crystal power accessory (CPA) is a small detachable board that connects to the back of the controller pod and is used to run the VLSiCE-96 emulator in the stand-alone mode. It is also used when running customer confidence tests. In the stand-alone mode, the user plug on the user cable is connected through the 68-pin PGA adaptor to the CPA. The CPA supplies clock and power. Stand-alone mode is used to test and debug software prior to the availability of hardware.

The optional multi-synchronous accessory can be used to connect the VLSiCE-96 emulator with up to 20 multi-ICE compatible emulators together for synchronous GO and BREAK emulation, and TRACE.

It can also be used with other debug equipment such as logic analyzers and oscilloscopes for synchronous GO, BREAK and TRACE.

The serial cable connects the host system to the controller pod. The serial cable has electrical specifications similiar to the RS-232C standard.

The power supply connects to the controller pod via the V_{CC} booster module and the DC power cable. There are several voltage options available for the power supply depending on switch settings on the back of the power supply.

A comprehensive set of documentation is included with the VLSiCE-96 emulator.

Figure 1 shows a drawing of the VLSiCE-96 emulator.

VLSiCE-96 EMULATOR FEATURES

The VLSiCE-96 emulator assists hardware and software design engineers in developing, debugging and testing designs incorporating the 8X9X family of microcontrollers. The following are some of the VLSiCE-96 features:

Emulation

Emulation is the controlled execution of the prototype software in the prototype hardware or in an artificial hardware environment that duplicates the microcontroller of the target system. With the VLSiCE-96 emulator, emulation is a transparent process that happens in real-time without sacrificing microcontroller resources. The execution of prototype software is facilitated through the VLSiCE-96 command language.

Memory Mapping

There are 64 Kbytes of zero-waitstate, high-speed mappable memory available. This memory space can be mapped to either the target system or to the on-board VLSiCE-96 memory space in 1 Kbyte blocks on 1 Kbyte boundaries. Mapping memory to the VLSiCE-96 emulator allows software development to proceed before prototype hardware is available. Memory mapping also gives the VLSiCE-96 emulator the capability to simulate the 8 Kbytes of (EP)ROM on those versions of the chip for code verification and validation.

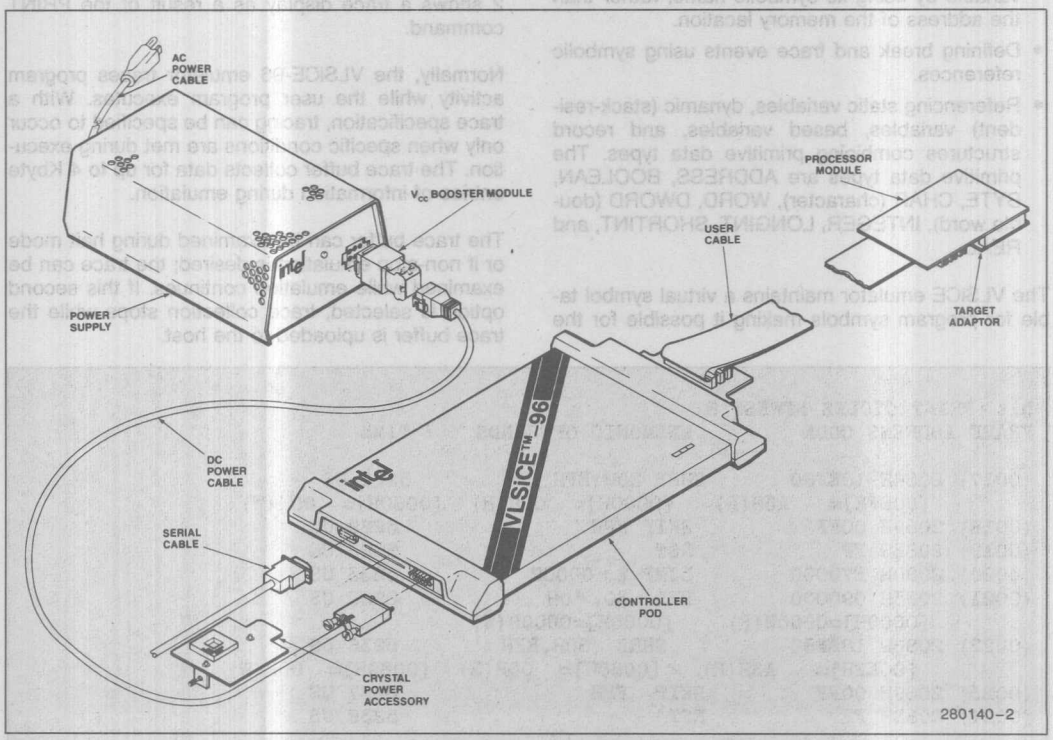


Figure 1. The VLSiCE™-96 Emulator

Memory Examination and Modification

The memory space for the 8X9X component and its target hardware is accessible through the emulator. The VLSiCE-96 software allows the component's special function registers to be accessed mnemonically (e.g. AD__RESULT, INT__MASK). A significant benefit to the VLSiCE-96 is its ability to read many of the write-only registers (e.g. IOC0, PWM__CONTROL) and to write many of the read-only registers (e.g. AD__RESULT, SBUFRX).

Data can be displayed or modified in several bases: hex, decimal, and binary, and in standard formats including: ASCII, real and integer. Program code can be disassembled and displayed as assembler mnemonics. It also can be modified with standard assembler statements.

Memory locations can also be examined or modified by their symbolic references. A symbolic reference is a procedure name, line number, or label in the user program that corresponds to a memory location.

Some typical symbolic functions include:

- Changing or inspecting the value of a program variable by using its symbolic name, rather than the address of the memory lacation.
- Defining break and trace events using symbolic references.
- Referencing static variables, dynamic (stack-resident) variables, based variables, and record structures combining primitive data types. The primitive data types are ADDRESS, BOOLEAN, BYTE, CHAR (character), WORD, DWORD (double word), INTEGER, LONGINT, SHORTINT, and REAL.

The VLSiCE emulator maintains a virtual symbol table for program symbols making it possible for the table to exist without fitting entirely into host RAM memory. The size of the virtual symbol table is constrained only by the capacity of the disk.

Breakpoint Specifications

Breakpoints allow halting of a user program in order to examine the effect of the program's execution on the target system. Breakpoints can be defined as execution addresses, data addresses and data values (both external and **internal RAM**), opcodes, selected bits of the PSW flag, and as 2 external inputs (SYNC0IN AND SYNC1IN). These breaks can be arranged to occur over a range of addresses and with up to 8 levels of arming and disarming. After a break the user program can resume execution from where it left off.

Trace Specifications

Tracing can be triggered with the same conditions set for breaking. The trace buffer is displayed as disassembled instructions, data fetches and stores, and with the timetag showing the relative time at which the program executed each instruction. Figure 2 shows a trace display as a result of the PRINT command.

Normally, the VLSiCE-96 emulator traces program activity while the user program executes. With a trace specification, tracing can be specified to occur only when specific conditions are met during execution. The trace buffer collects data for up to 4 Kbyte entries of information during emulation.

The trace buffer can be examined during halt mode or if non-stop emulation is desired; the trace can be examined while emulation continues. If this second option is selected, trace collection stops while the trace buffer is uploaded to the host.

```
hlt>PRINT CYCLES NEWEST 8
FRAME ADDRESS CODE        MNEMONIC OPERANDS      TIME

(0017) 2086H 18EF80       SHRB 80H,EFH           5221 US
       [00EFH]=   A3H(R)    [0080H]=  00H(R)  [0080H]=  00H(W)
(0018) 2089H 00FF         SKIP FFH               5222 US
(0019) 208BH FF           RST                    5223 US
(0020) 2080H E70000       LJMP $+0003H           5233 US
(0021) 2083H 090000       SHL  R0,#0H            5225 US
       [0000H]=0000H(R)     [0000H]=0000H(W)
(0022) 2086H 18EF80         SHRB  80H,EFH        5236 US
       [00EFH]=   A3H(R)    [0080H]=  00H(R)  [0080H]=  00H(W)
(0023) 2089H 00FF         SKIP  FFH              5237 US
(0024) 208BH FF           RST                    5238 US
```

Figure 2. The Trace Buffer Display

Arming and Triggering

The VLSiCE-96 command language allows specification of complex events with up to 8 states, each with several conditions. For example, a specification can be made that causes a break to occur when a variable is written only within a certain procedure. The execution of the procedure is the arm condition and the variable modification is the break condition. The arm condition is an optional part of a break/trace sequence in the VLSiCE emulator. A set of arm conditions can be used to ensure that a break is not possible until all required qualifying conditions are satisfied.

Procedures

Debugging procedures (PROCS) are a user-named group of VLSiCE commands that are executed sequentially. Procs can simulate missing hardware or software, collect debug information, and make troubleshooting decisions. They can be copied to text files on disk, then included from the file into the command sequence in later test sessions.

Procedures can also serve as programmable diagnostics, implementing new emulator commands for special purpose.

FASTBREAKS

Fastbreaks make it possible to examine and modify memory without halting emulation. The commands that can be executed are simple one-access functions, such as, WORD 1FH or IOS0. When enabled, fastbreaks occur whenever a memory access is made.

Breakpoints and tracepoints can be re-specified during emulation with fastbreaks enabled.

While emulation does not halt during fastbreaks, a delay in code execution occurs when a fastbreak is requested. In most cases, this latency in code execution is less than 150 μs.

Interrupts During Interrogation (IDI) Mode

The VLSiCE-96 software can service and record interrupts even though emulation has been halted (interrogation mode). In the special mode designated as IDI mode, hardware interrupts can be serviced while the emulator is being interrogated. Use of this mode is determined by the setting of a VLSiCE-96 pseudo-variable (IDI_PC). After breaking from emulation or fastbreaks mode, whenever an interrupt occurs, the processor jumps to the appropriate vector and executes the interrupt service routine.

The setting of another VLSiCE-96 pseudo-variable (INT_REC_EN) allows the recording of interrupts but not the servicing of interrupts, during halt mode. If the pseudo-variable is set to TRUE, all interrupts are recorded in the INT_PENDING register, and serviced when the emulator re-enters emulation.

Dynamic Tracing

The trace buffer can be accessed in two ways, dynamically during emulation and statically after emulation halts. While dynamically tracing, any form of the PRINT command can be entered and the specified portion of the trace buffer is displayed. This allows real-time display of processor activity. Displaying the trace buffer during emulation stops collection of trace and some trace information can be lost, but emulation is unaffected.

On-Line Syntax Guide

A special syntax guide called the Integrated Command Directory (ICD), at the bottom of the display screen, aids in creating syntactically correct command lines. Figure 3 shows an example of the ICD for the GO command.

HELP

This feature provides assistance with the emulator commands through the host terminal. HELP is available for most of the commands. Figure 4 shows help for one of the commands.

Multi-Synchronous Operation

The VLSiCE-96 emulator can run with other emulators, and lab equipment such as logic analyzers or oscilloscopes. VLSiCE-96 emulators can be daisy-chained together in a network to work simultaneously to test a prototype system. The multi-synchronous operation is facilitated by the optional multi-synchronous accessory.

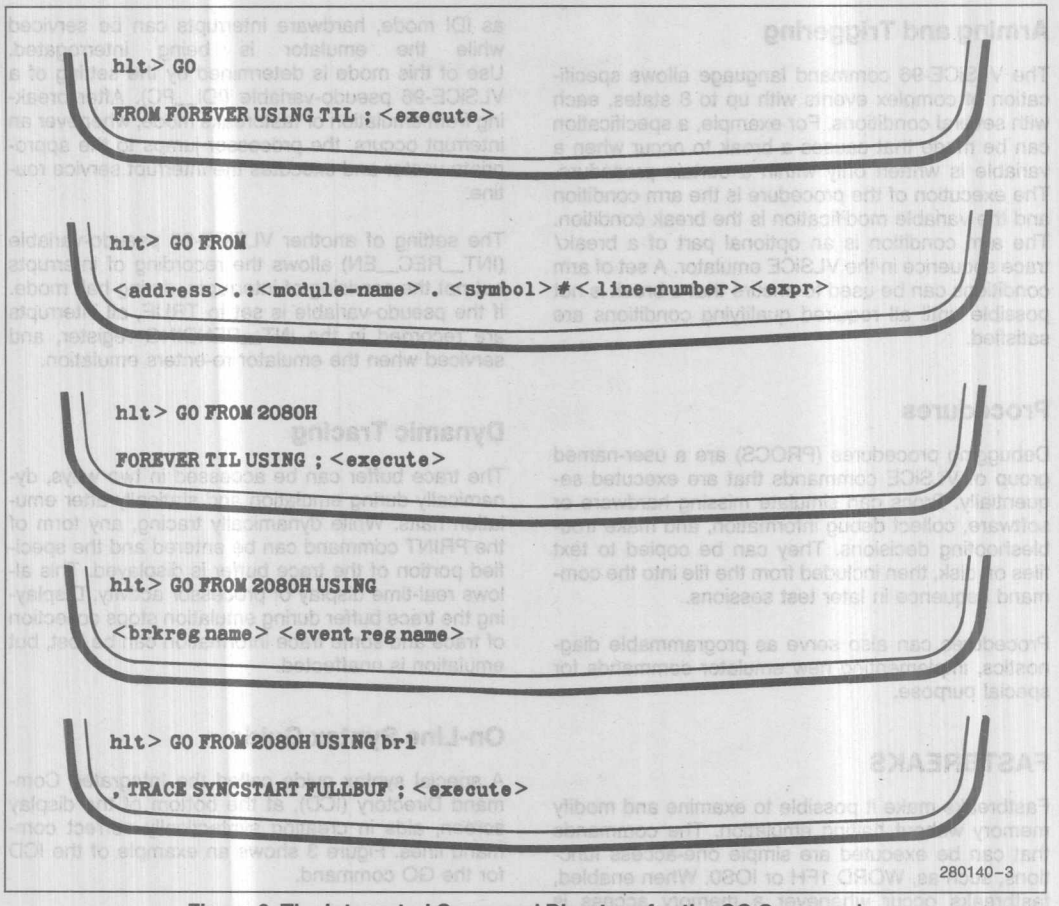


Figure 3. The Integrated Command Directory for the GO Command

DESIGN CONSIDERATIONS

There are design considerations to be aware of before designing with the VLSiCE-96 emulator.

Electrical Considerations

The user pin timings, thresholds, and loadings are identical to the 8096 component except the RESET and CLKOUT pins have an additional loading of 1 μA and 10 pF. The Non-Maskable Interrupt (NMI) is not supported.

	Min.	Max.
Clock Frequency	6 MHz	10 MHz
V_{CC}	Emulator does not require system power to operate.	
I_{CC}		0 mA

Mechanical Considerations

The user plug is on the end of a three foot (1m) flexible cable. Adequate spacing must be provided on the target system to allow the emulation processor board and user plug to be inserted into the target system.

The height of the user plug should be considered for multiple board system prototypes that need to be debugged and tested. Be sure that the space between the boards is greater than 1½" (4 cm) to allow for the user plug.

Figure 5 shows the user plug dimensions. The user plug comprises the emulator processor board and the 68-pin or 48-pin adaptor. In the figure, please note the location of pin 1 on each adaptor.

```
hlt> HELP ASM
The ASM command displays or modifies memory as 8096 mnemonics.
The syntax is:

 ASM <asm-spec>

 <asm-spec> ::=<partition> [='<asm96-inst>'[.'<asm96-inst>']*]
              | <address> = <cr>

where:

  <partition>  specifies the area of memory to be displayed or modified.

  <asm96-inst> specifies the 8096 assembly instructions to be assembled.

  <address>    is any valid 8096 address.

  <cr>         indicates a carriage-return.

The "ASM <address> =" syntax puts the user in line-mode, displaying the
current address at which the assembly instruction will be placed and not
requiring the quotes around the instructions.

Please see the VLSiCE-96 User's Guide for more detailed information.
```

Figure 4. HELP Screen

Other Considerations

- The non-maskable interrupt (NMI) is not supported.
- The counters for the pulse width modulator (PWM) and hardware Timer1 can be out of sync if either are disabled during interrogation. Synchronize them by resetting the emulator.
- The Zero flag is always cleared in the SUBC instruction. Therefore, the relational operators <= and > for LONG variables in C96 V1.0 and LONGINT variables in PL/M-96 V1.1, work incorrectly. These languages have been tailored for the 8X9X-90 microcontroller which either sets or resets the Zero flag in the SUBC instruction.

If there is a memory-resident program that is permanent on the PC, use of the DOS shell escape may corrupt the VLSiCE-986 software. To insure reliability, do not use the system escape on host systems that have permanent memory-resident programs.

The VLSiCE-96 emulator has some properties that are inherent in the 8X9XBH component. These are:

- Neither the source nor the destination address of the Multiply or Divide instructions can be a writable special function register.
- The special function registers, except R0, may not be used as base or index registers for indexed or indirect instructions.
- Several of the special function registers can only be accessed as words, while others only as bytes. These restrictions are listed in the 8096 User's Manual.
- The sticky flag is not affected when a skip by 0 is executed.
- To emulate the 8X9X-90 microcontroller, memory location 2018H in both target system emulator mapped memory should be 0FFH.
- The JBS and JBC instructions cannot be used directly on Port 2.1.

SPECIFICATIONS

Host Requirements

An IBM PC XT or PC AT with 512 Kbytes RAM and hard disk. Intel recommends and IBM PC AT with 640 Kbytes of RAM, one floppy drive and one hard disk, running PC-DOS 3.1 or later.

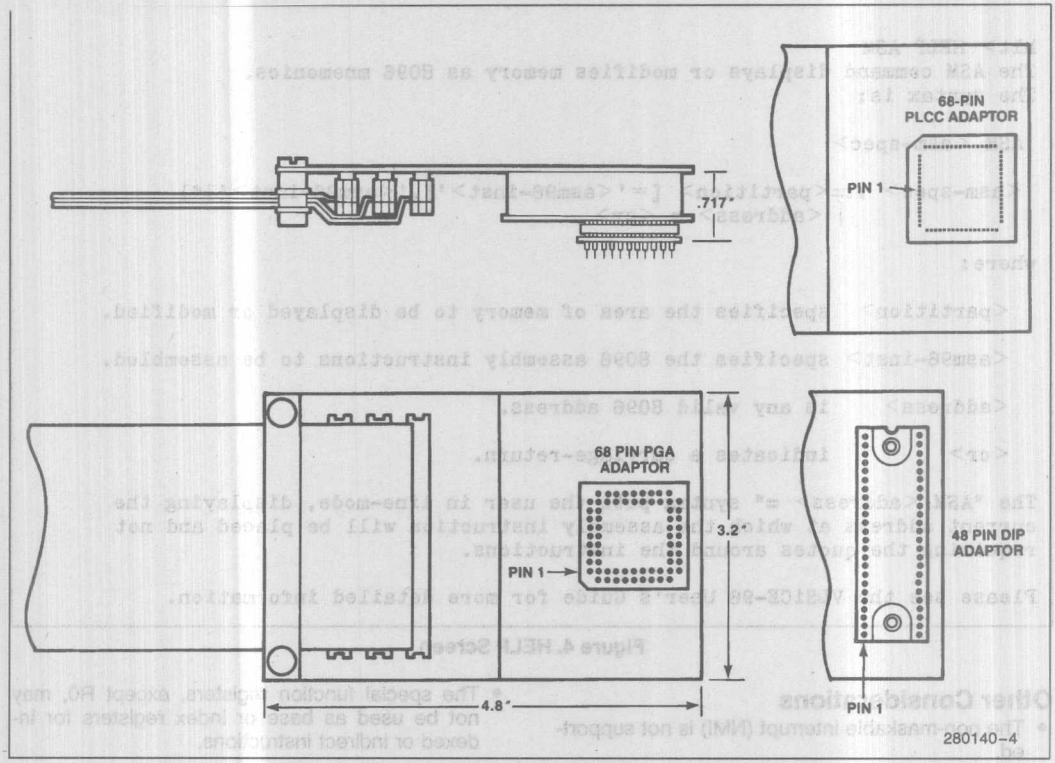


Figure 5. Dimensions for the Emulator Processor Board and Adaptors

An Intellec® Microcomputer Development System, Series III or Series IV, running under ISIS or iNDX, with at least 512 Kbytes of application memory resident, with dual floppy or one hard disk and one floppy drive required.

VLSiCE-96 Software Package

VLSiCE-96 emulator software

VLSiCE-96 confidence tests

VLSiCE-96 tutorial software

System Performance

Mappable zero wait-state (up to 10 MHz).	Min 0 Kbytes Max 64 Kbytes	Mappable to user memory or ICE memory in 1K blocks on 1K boundaries.
Trace Buffer	4 Kbytes × 48 bits	
Virtual Symbol Table	A maximum of 61 Kbytes of host memory space is available for the virtual symbol table (VST). The rest of the VST resides on disk and is paged in and out of host memory as needed.	

Physical Characteristics

Controller Pod

Width: 8¼" (21 cm)

Height: 1½" (4 cm)

Depth: 13½ (34 cm)

Weight: 4 lbs (2 kg)

Power Supply

Width: 7⅝″ (18 cm)
Height: 4″ (10 cm)
Depth: 11″ (28 cm)
Weight: 15 lbs (7 kg)

User Cable

3′ (1 m)

Serial Cable

12′ (4 m)

Electrical Characteristics

Power Supply

100V–120V or 200V–240V (selectable)
50 Hz–60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

Environmental Characteristics

Operating Temperature: 0°C to +40°C (+32°F to +104°F)

Operating Humidity: Maximum of 85% relative humidity, non-condensing

DOCUMENTATION

VLSiCE-96 In-Circuit Emulator User's Guide, order number 165814

VLSiCE-96 In-Circuit Emulator Pocket Reference, order number 165815

VLSiCE-96 In-Circuit Emulator Installation Supplement, order number 166477

VLSiCE-96 Emulator Tutorial Guide, order number 165816

Debug Editor User's Guide, order 167098

ORDERING INFORMATION

Emulator Hardware and Software

Order Code	Description
V096KITA	VLSiCE-96 power supply and cable, emulation base, user cable, CPA, serial cables for PC AT and PC XT, 68-pin PGA target adaptor, ASM-96, AEDIT text editor. Host, probe, diagnostic and tutorial software is on 5¼″ media for DOS hosts running DOS V3.0 or greater. [Requires software license.]
V096KITD	Same as V096KITAD without ASM-96 and AEDIT text editor.
V096KITAS	VLSiCE-96 power supply and cable, emulation base, user cable, serial cable, CPA, 68-pin PGA target adaptor, ASM-96, and AEDIT text editor. Host, probe, diagnostic and tutorial software is on 8″ single density and 8″ double density media for hosting on an Intel Series III, and 5¼″ media for Series IV [Requires software license.]
V096KITS	Same as V096KITAS without ASM-96 and AEDIT text editor.

Target Adaptor

Order Code	Description
TA096E	Optional 68-pin PLCC Adaptor board
TA096B	Optional 48-pin DIP Target Adaptor board.

Multi-Synchronous Accessory

Order Code	Description
MSA96	Optional Multi-Synchronous Accessory for multi-ICE capability.

Software Only

Order Code	Description
SA096D	Software for host, probe, diagnostic and tutorial on 5¼" media for use with the PC AT and PC XT under PC-DOS V3.0 or greater. [Requires software license.]
SA096SD	Software for host, probe, diagnostic and tutorial on 5¼" media for use with the PC AT and PC XT under PC-DOS V3.0 or greater. [Requires software license.]
SA096S	Software for host, probe, diagnostic and tutorial on 8" single density and 8" double-density media for use with a Series III, and 5¼" media for use with a Series IV. [Requires software license.]

Other Useful Intel 8X9X Debug and Development Support Products

Order Code	Description
I86ASM96	Consists of the ASM 96 macro assembler that translates symbolic assembly language mnemonics into relocatable object code, and the RL96 linker and relocator program that links modules generated by ASM 96 and PL/M 96 and locates the linked object modules to absolute memory locations. System requirements and Intellec System running iNDX.
I86PLM96	Consists of the PL/M 96 compiler that provides high level programming language support, the LIB 96 utility that creates and maintains libraries of software object modules, the FPAL96 floating point arithmetic library, and the RL96 linker and relocator program that links modules generated by ASM 96 and PL/M 96 and locates the linked object modules to absolute memory locations. System requirements and Intellec System running iNDX.
D86ASM96NL	ASM/R&L 96 for PC-DOS. It contains a macro assembler, a linker/locator utility, a floating point utility and a librarian. System requirements are an IBM PC AT or PC XT with 512 Kbytes of RAM and PC-DOS 3.0 or greater.
D86PLM96NL	PL/M 96 and R&L for PC-DOS. It contains a compiler, a linker/locator utility, a floating point utility and a librarian. System requirements are an IBM PC AT or PC XT with 512 Kbytes of RAM and PC-DOS 3.0 or greater.
D86C96NL	C96 and R&L for PC-DOS. Contains a compiler linker/locator utility, and all standard C libraries including STDIO. System requirements are an IBM PC AT or PC XT with 512 Kbytes of RAM and PC-DOS 3.0 or greater.
pSBE96SKIT	iSBE-96 single board emulator for use with the Series III/IV development systems. The kit contains: iSBE-96 single board emulator iSBE-96 Series III/IV upgrade kit (cables and software needed to run on Intel Hosts).
pSBE96DKIT	iSBE-96 single board emulator for use with the IBM PC AT and PC XT computer systems. The kit contains: iSBE-96 single board emulator 8096 software support package for PC-DOS. iSBE-96 DOS upgrade kit (cables and software needed to run on the IBM PC AT or PC XT).

Running the iSBE-96 emulator on the Series II and iPDS system requires software from:

U.S. Software Corporation
5470 N.W. Innisbrook
Portland, OR 97229
Phone: 503-645-5043
International Telex: 4993875

ICE™-5100/452 IN-CIRCUIT EMULATOR FOR THE UPI™-452 FAMILY OF PROGRAMMABLE I/O PROCESSORS

- **Precise, Full-Speed, Real-Time Emulation of the UPI™-452 Family of I/O Processors**
- **64 KB of Mappable High-Speed Emulation Memory**
- **254 24-Bit Frames of Trace Memory (16 Bits Trace Program Execution Addresses and 8 Bits Trace External Events)**
- **Serial Link to the IBM* PC AT, PC XT (and DOS compatibles), and the Intellec® Series III/IV**
- **ASM-51 and PL/M-51 Language Support**
- **Full Emulation and Debug Support for the FIFO Buffer**
- **Built-In CRT-Oriented Text Oriented**
- **Symbolic Debugging Enables Access to Memory Locations and Program Variables**
- **Four Address Breakpoints with In-Range, Out-of-Range, and Page Breaks**
- **Equipped with the Integrated Command Directory (ICD™) that Provides:**
  - **— On-Line Help**
  - **— Syntax Guidance and Checking**
  - **— Dynamic Command Entry**
  - **— Error Checking**
  - **— Command Recall**
- **On-Line Disassembler and Single-Line Assembler to Help with Code Patching**

The ICE™-5100/452 in-circuit emulator is a high-level, interactive debugger that is used to develop and test the hardware and software of a target system based on the UPI™-452 family of I/O processors. The ICE-5100/452 emulator can be serially linked to an Intellec® Series III/IV or an IBM PC AT or PC XT. The emulator can communicate with the host system at standard baud rates up to 19.2K.



280338-1

August 1987
Order Number: 280338-003

PRODUCT OVERVIEW

The ICE-5100/452 emulator provides full emulation support for the UPI-452 family of I/O processors. The UPI-452 family consists of the 83452, 87452, and the 80452.

The ICE-5100/452 emulator enables hardware and software development to proceed simultaneously. With the ICE-5100/452 emulator, prototype hardware can be added to the system as it is designed and software can be developed prior to the completion of the hardware prototype. Software and hardware integration can occur while the product is being developed.

The ICE-5100/452 emulator assists four stages of development:

- Software debugging
- Hardware debugging
- System integration
- System test

SOFTWARE DEBUGGING

The ICE-5100/452 emulator can be operated without being connected to the target system and before any of the user's hardware is available (provided external data RAM is not needed). In this stand-alone mode, the ICE-5100/452 emulator can be used to facilitate program development.

HARDWARE DEBUGGING

The ICE-5100/452 emulator's AC/DC parametric characteristics match the microcontroller's. The emulator's full-speed operation makes it a valuable tool for debugging hardware, including time-critical serial port, timer, and external interrupt interfaces.

SYSTEM INTEGRATION

Integration of software and hardware can begin when the emulator is plugged into the microcontroller socket of the prototype system hardware. Hardware can be added, modified, and tested immediately. As each section of the user's hardware is completed, it can be added to the prototype. Thus, the hardware and software can be system tested in real-time operation as each section becomes available.

SYSTEM TEST

When the prototype is complete, it is tested with the final version of the system software. The ICE-5100/452 emulator is then used for real-time emulation of the microcontroller to debug the system as a completed unit.

The final product verification test can be performed using the ROM or EPROM version of the microcontroller. Thus, the ICE-5100/452 emulator provides the ability to debug a prototype or production system at any stage in its development without introducing extraneous hardware or software test tools.

PHYSICAL DESCRIPTION

The ICE-5100/452 emulator consists of the following components (see Figure 1):

- Power supply
- AC and DC power cables
- Controller pod
- Serial cable (host-specific)
- User probe assembly (consisting of the processor module and the user cable)
- Crystal power accessory (CPA)
- 68-pin PGA target adaptor
- Clips assembly
- Software (includes the ICE-5100/452 emulator software, diagnostic software, and a tutorial)

The controller pod contains 64 KB of emulation memory, 254- by 24-bit frames of trace memory, and the control processor. In addition, the controller pod houses a BNC connector that can be used to connect up to 10 multi-ICE compatible emulators for synchronous starting and stopping of emulation.

The serial cable connects the host system to the controller pod. The serial cable supports a subset of the RS-232C signals.

The user probe assembly consists of a user cable and a processor module. The processor module houses the emulation processor and the interface logic. The target adaptor connects to the processor module and provides an electrical and mechanical interface to the target microcontroller socket.

The crystal power accessory (CPA) is a small, detachable board that connects to the controller pod and enables the ICE-5100/452 emulator to run in stand-alone mode. The target adaptor plugs into the socket on the CPA; the CPA then supplies clock and power to the user probe.

The clips assembly enables the user to trace external events. Eight bits of data are gathered on the rising edge of $\overline{\text{PSEN}}$ during opcode fetches. The clips information can be displayed using the CLIPS option with the PRINT command. Trace qualification

input and output lines are also provided on the clips pod for connection to test equipment.

The ICE-5100/452 emulator software supports mnemonics, object file formats, and symbolic references generated by Intel's ASM-51 and PL/M-51 programming languages. Along with the ICE-5100/452 emulator software is a customer confidence test disk with diagnostic routines that check the operation of the hardware.

The on-line tutorial is written in the ICE-5100 command language. Thus, the user is able to interact with and use the ICE-5100/452 emulator while executing the tutorial.

A comprehensive set of documentation is provided with the ICE-5100/452 emulator.

ICE™-5100/452 EMULATOR FEATURES

The ICE-5100/452 emulator has been created to assist a product designer in developing, debugging and testing designs incorporating the UPI-452 family of I/O processors. The following sections detail some of the ICE-5100/452 emulator features.

EMULATION

Emulation is the controlled execution of the user's software in the target hardware or in an artificial hardware environment that duplicates the microcontroller of the target system. Emulation is a transparent process that happens in real-time. The execution of the user software is facilitated with the ICE-5100/452 command language.

MEMORY MAPPING

The memory space for the 452 microcontroller and its target hardware is fully accessible through the emulator. The ICE-5100/452 emulator refers to four physically distinct memory spaces, as follows:

- CODE — references program memory
- IDATA — references internal data memory
- RDATA — references special function register memory
- XDATA — references external data memory

ICE-5100/452 emulator commands that access memory use one of the special prefixes (e.g., CODE) to specify the memory space.

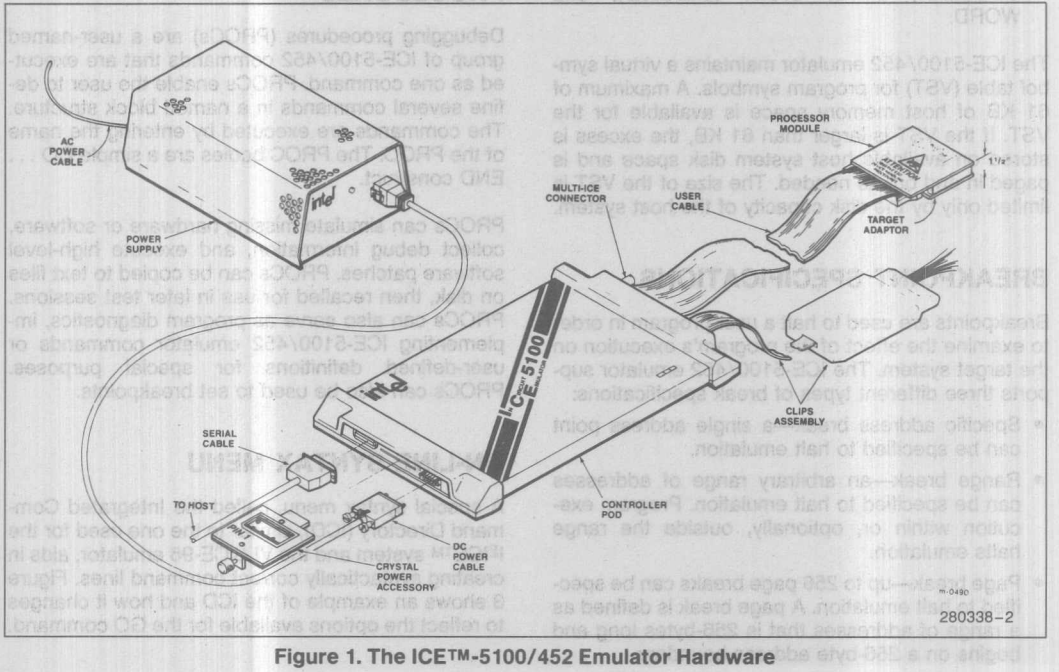


Figure 1. The ICE™-5100/452 Emulator Hardware

The microcontroller's special function registers and register bits can be accessed mnemonically (e.g., DPL, TCON, CY) with the ICE-5100/452 emulator software.

Data can be displayed or modified in one of three bases: hexadecimal, decimal, and binary. Data can also be displayed or modified in one of two formats: ASCII and unsigned integer. Program code can be disassembled and displayed as ASM-51 assembler mnemonics. Code can be modified with standard ASM-51 statements using the built-in single-line assembler.

Symbolic references can be used to specify memory locations. A symbolic reference is a procedure name, line number, program variable, or label in the user program that corresponds to a location.

Some typical symbolic functions include:

- Changing or inspecting the value of a program variable by using its symbolic name to access the memory location.
- Defining break and trace events using symbolic references.
- Referencing variables as primitive data types. The primitive data types are ADDRESS, BIT, BOOLEAN, BYTE, CHAR (character), and WORD.

The ICE-5100/452 emulator maintains a virtual symbol table (VST) for program symbols. A maximum of 61 KB of host memory space is available for the VST. If the VST is larger than 61 KB, the excess is stored on available host system disk space and is paged in and out as needed. The size of the VST is limited only by the disk capacity of the host system.

BREAKPOINT SPECIFICATIONS

Breakpoints are used to halt a user program in order to examine the effect of the program's execution on the target system. The ICE-5100/452 emulator supports three different types of break specifications:

- Specific address break—a single address point can be specified to halt emulation.
- Range break—an arbitrary range of addresses can be specified to halt emulation. Program execution within or, optionally, outside the range halts emulation.
- Page break—up to 256 page breaks can be specified to halt emulation. A page break is defined as a range of addresses that is 256-bytes long and begins on a 256-byte address boundary.

Break registers are user-defined debug definitions used to create and store breakpoint definitions. Break registers can contain multiple breakpoint definitions and can optionally call debug procedures when emulation halts.

TRACE SPECIFICATIONS

Tracing can be triggered using specifications similar to those used for breaking. Normally, the ICE-5100/452 emulator traces program activity while the user program is executing. With a trace specification, tracing can be triggered to occur only when specific conditions are met during execution. Up to 254 24-bit frames of trace information are collected in a buffer during emulation. Sixteen of the 24 bits trace instruction execution addresses, and 8 bits capture external events (CLIPS).

The trace buffer display is similar to an ASM-51 program listing as shown in Figure 2. The PRINT command enables the user to selectively display the contents of the trace buffer. The user has the option of displaying the clips information as well as disassembled instructions.

PROCEDURES

Debugging procedures (PROCs) are a user-named group of ICE-5100/452 commands that are executed as one command. PROCs enable the user to define several commands in a named block structure. The commands are executed by entering the name of the PROC. The PROC bodies are a simple DO . . . END construct.

PROCs can simulate missing hardware or software, collect debug information, and execute high-level software patches. PROCs can be copied to text files on disk, then recalled for use in later test sessions. PROCs can also serve as program diagnostics, implementing ICE-5100/452 emulator commands or user-defined definitions for special purposes. PROCs can also be used to set breakpoints.

ON-LINE SYNTAX MENU

A special syntax menu, called the Integrated Command Directory (ICD), similar to the one used for the I²ICE™ system and the VLSiCE-96 emulator, aids in creating syntactically correct command lines. Figure 3 shows an example of the ICD and how it changes to reflect the options available for the GO command.

HELP

The HELP command provides ICE-5100/452 emulation command assistance via the host system terminal. ON-line HELP is available for the ICE-5100/452 emulator commands shown in Figure 4.

DESIGN CONSIDERATIONS

The height of the processor module and the target adaptor need to be considered for target systems. Allow at least 1½ inches (3.8 cm) of space to fit the processor module and target adaptor. Figure 5 shows the dimensions of the processor module.

```
hlt>PRINT NEWEST 4          /* Print newest four intruction in
                               buffer */
FRAME       ADDRESS    CODE        INSTRUCTION
(028)       300A       C02A        PUSH   2AH
(030)       300C       2532        ADD    A ,32H
(032)       300E       F52A        MOV    2AH ,A
(034)       3010       B53210      CJNE   A ,32H ,, + 10H
hlt>
hlt>PRINT CLIPS OLDEST 2  /* Buffer display showing clips */
FRAME       ADDRESS    CODE        INSTRUCTION          CLIPS (76543210)
(000)       300A       C02A        PUSH   2AH                01110011
(001)       300C       2532        ADD    A ,32H             11110101
hlt>
```

280338-3

Figure 2. Selected Trace Buffer Displays

```
hlt> GO
FROM ARM FOREVER TIL USING TRACE ; <execute>

hlt> GO FROM
<expr>

hlt> GO FROM 13H
<operator> ARM FOREVER TIL USING TRACE ; <EXECUTE>

hlt> GO FROM 13H USING
BRKREG <brkreg name>

hlt> GO FROM 13H USING brl
, TRACE ; <execute>

hlt> GO FROM 13H USING brl TRACE
<expr> OUTSIDE PAGE FROM TIL <trcreg name> ; <execute>

hlt> GO FROM   13H USING brl TRACE tracelt
; <execute>
```

280338-4

Figure 3. The Integrated Command Directory for the GO Command

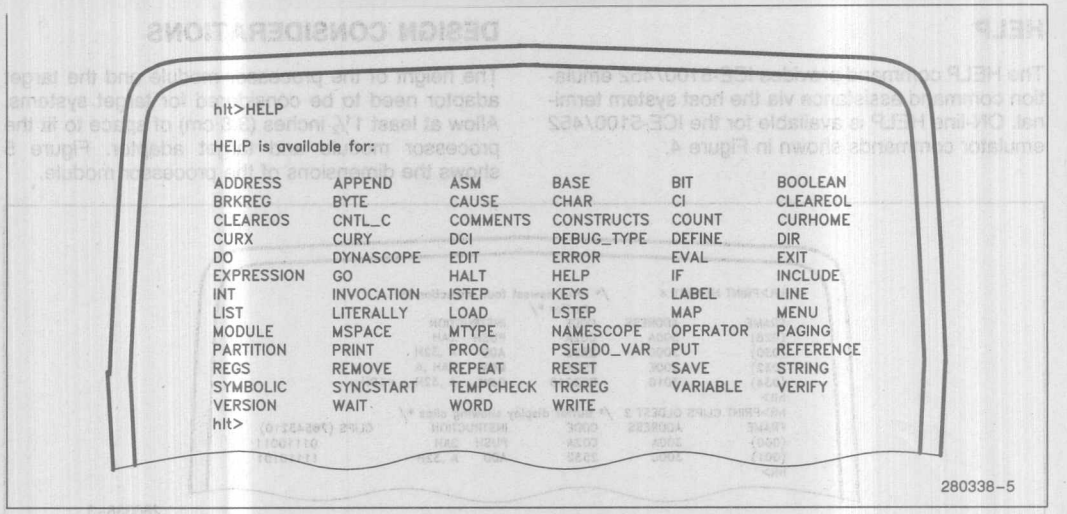


Figure 4. HELP Menu

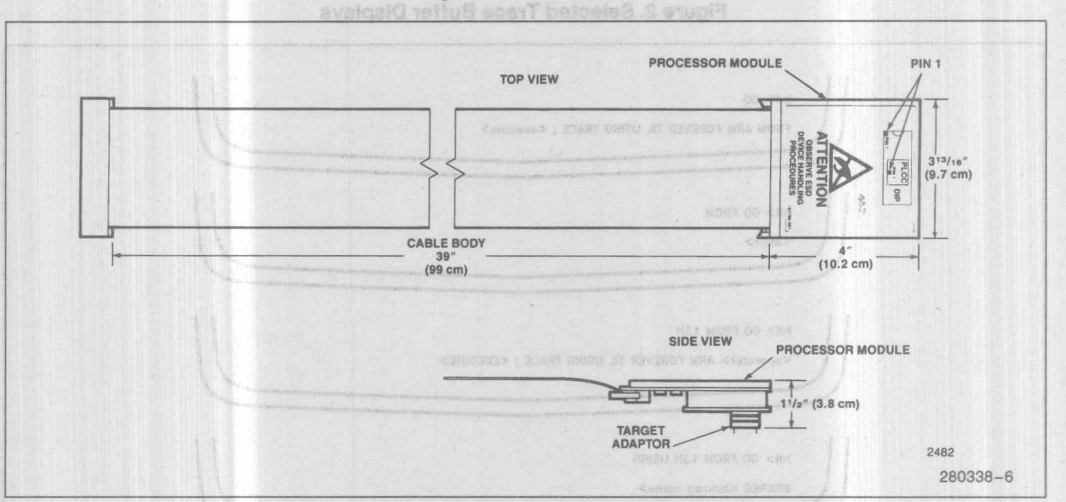


Figure 5. Processor Module Dimensions

ELECTRICAL CONSIDERATIONS

The emulation processor's user-pin timings and loadings are identical to the 452 component, except as follows:

- Up to 25 pF of additional pin capacitance is contributed by the processor module and target adaptor assemblies.

HOST REQUIREMENTS

- IBM PC AT or PC XT (or DOS compatible) with 512 KB of available RAM and a hard disk running under the DOS 3.0 (or later) operating system.
- Intellec Series III/IV microcomputer development system running the ISIS or iNDX operating system respectively, with at least 512 KB of application memory resident.
- Disk drives—dual floppy or one hard disk and one floppy drive required.

ICE™—452 EMULATOR SOFTWARE PACKAGE

- ICE-5100/452 emulator software
- ICE-5100/452 confidence tests
- ICE-5100 tutorial software

EMULATOR PERFORMANCE

MEMORY

Mappable full-speed emulation code memory	64 KB	Mappable to user or ICE-5100/452 emulator memory in 4 KB blocks on 4 KB boundaries.
Trace memory		254 x 24 bit frames.
Virtual Symbol Table		A maximum of 61 KB of host memory space is available for the virtual symbol table (VST). The rest of the VST resides on disk and is paged in and out as needed.

PHYSICAL CHARACTERISTICS

CONTROLLER POD

Width: 8¼" (21 cm)
Height: 1½" (3.8 cm)
Depth: 13½" (34.3 cm)
Weight: 4 lbs (1.85 kg)

USER CABLE

The user cable is 3 feet (approximately 1m).

PROCESSOR MODULE

(With the target adaptor attached.)
Width: 3 13/16" (9.7 cm)
Height: 4" (10.2 cm)
Depth: 1½" (3.8 cm)

POWER SUPPLY

Width: 7⅝" (18.1 cm)
Height: 4" (10.06 cm)
Depth: 11" (27.97 cm)
Weight: 15 lbs (6.1 kg)

SERIAL CABLE

The serial cable is 12 feet (3.6 m).

ELECTRICAL CHARACTERISTICS

POWER SUPPLY

100V–120V or 200V–240V (selectable)
50 Hz–60 Hz
2 amps (AC max)[9] 120V
1 amp (AC max)[9] 240V

ENVIRONMENTAL CHARACTERISTICS

Operating temperature: +10°C to +40°C (50°F to 104°F)
Operating humidity: Maximum of 85% relative humidity, non-condensing

ORDERING INFORMATION

EMULATOR HARDWARE AND SOFTWARE

Order Code	Description
I452KITAD	This kit contains the ICE-5100/452 user probe assembly, power supply and cables, serial cables, target adaptor, CPA, ICE-5100 controller pod, software, and documentation for use with an IBM PC AT or PC XT. The kit also includes the 8051 Software Development Package and the AEDIT text editor for use on DOS systems. (Requires software license.)
I452KITD	This kit is the same as the I452KITAD kit excluding the 8051 Software Development Package and the AEDIT text editor. (Requires software license.)
I452KITAS	This kit contains the ICE-5100/452 user probe assembly, power supply and cables, serial cables, target adaptor, CPA, ICE-5100 controller pod, software, and documentation for use with Intel hosts (Series III/IV). The kit also includes the 8051 Software Development Package and the AEDIT text editor for use on the Series III/IV. (Requires software license.)
I452KITS	This kit is the same as the I452KITAS kit excluding the 8051 Software Development Package and the AEDIT text editor. (Requires software license.)

SOFTWARE ONLY

Order Code	Description
SA452D	This kit contains the host, probe, diagnostic, and tutorial software on 5¼" disks for use on an IBM PC AT or PC XT (requires DOS 3.0 or later). (Requires software license.)
SA452S	This kit contains the host, probe, diagnostic and tutorial software on 8" disks (both single-density and double-density) for use on a Series III, and on 5¼" disks for use on a Series IV. (Requires software license.)

OTHER USEFUL DEBUG AND DEVELOPMENT SUPPORT PRODUCTS

Order Code	Description
D86ASM51	**8051 Software Development Package** (DOS version)—Consists of the ASM-51 macro assembler which gives symbolic access to 8051 hardware features; the RL51 linker and relocator program that links modules generated by ASM-51; CONV51 which enables software written for the MCS-48 family to be upgraded to run on the 8051, and the LIB51 Librarian which programmers can use to create and maintain libraries of software object modules. Use with the DOS operating system (version 3.0 or later).
D86PLM51	**PL/M-51 Software Package** (DOS version)—Consists of the PL/M compiler which provides high-level programming language support; the LIB51 utility that creates and maintains libraries of software object modules, and the RL51 linker and relocator program that links modules generated by ASM-51 and PL/M-51 and locates the linked object modules to absolute memory locations. Use with the DOS operating system (version 3.0 or later).
I86ASM51	**8051 Software Development Package** (ISIS version)—Same as the D86ASM51 package except this one is for use with the Series III.
I86PLM51	**PL/M Software Package**—Same as the D86PLM51 package except this one is for use with the Series III and the Series IV.
D86EDINL	AEDIT text editor for use with the DOS operating system.

ICE™-5100/044 In-Circuit Emulator for the RUPI™-44 Family

- **Precise, Full-Speed, Real-Time Emulation of the RUPI™-44 Family of Peripherals**
- **64 KB of Mappable High-Speed Emulation Memory**
- **254 24-bit Frames of Trace Memory (16 Bits Trace Program Execution Addresses and 8 Bits Trace Eternal Events)**
- **Serial Link to Intel Series III/IV or IBM* PC AT or PC XT (and PC DOS Compatibles)**
- **ASM-51 and PL/M-51 Language Support**
- **Built-in CRT-Oriented Text Editor**
- **Symbolic Debugging Enables Access to Memory Locations and Program Variables**
- **Four Address Breakpoints Plus In-Range, Out-of-Range, and Page Breaks**
- **Equipped with the Integrated Command Directory (ICD™) That Provides**
  — On-Line Help
  — Syntax Guidance and Checking
  — Command Recall
- **On-Line Disassembler and Single-Line Assembler to Help with Code Patching**
- **Provides an Ideal Environment for Debugging BITBUS™ Applications Code**

The ICE™-5100/044 in-circuit emulator is a high-level, interactive debugger that is used to develop and test the hardware and software of a target system based on the RUPI™-44 family of peripherals. The ICE-5100/044 emulator can be serially linked to an Intellec® Series III/IV or an IBM PC AT or PC XT. The emulator can communicate with the host system at standard baud rates up to 19.2K. The design of the emulator supports all of the RUPI-44 components at speeds up to and including 12 MHz.

*IBM is a registered trademark of International Business Machines Corporation. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.



280325-1

November 1986
Order Number: 280325-001

PRODUCT OVERVIEW

The ICE-5100/044 emulator provides full emulation support for the RUPI-44 family of peripherals, including 8044-based BITBUS™ board products. The RUPI-44 family consists of the 8044, the 8744, and the 8344.

The ICE-5100/044 emulator enables hardware and software development to proceed simultaneously. With the ICE-5100/044, prototype hardware can be added to the system as it is designed and software can be developed prior to the completion of the hardware prototype. Software and hardware integration can occur while the product is being developed.

The ICE-5100/044 emulator assists four stages of development:

- Software debugging
- Hardware debugging
- System integration
- System test

Software Debugging

The ICE-5100/044 emulator can be operated without being connected to the target system and before any of the user's hardware is available (provided external data RAM is not needed). In this stand-alone mode, the ICE-5100/044 emulator can be used to facilitate program development.

Hardware Debugging

The ICE-5100/044 emulator's AC/DC parametric characteristics match the microcontroller's. The emulator's full-speed operation makes it a valuable tool for debugging hardware, including time-critical serial port, timer, and external interrupt interfaces.

System Integration

Integration of software and hardware can begin when the emulator is plugged into the microcontroller socket of the prototype system hardware. Hardware can be added, modified, and tested immediately. As each section of the user's hardware is completed, it can be added to the prototype. Thus, the hardware and software can be system tested in real-time operation as each section becomes available.

System Test

When the prototype is complete, it is tested with the final version of the system software. The ICE-5100/044 emulator is then used for real-time emulation of the microcontroller to debug the system as a completed unit.

The final product verfication test can be performed using the ROM or EPROM version of the microcontroller. Thus, the ICE-5100/044 emulator provides the ability to debug a prototype or production system at any stage in its development without introducing extraneous hardware or software test tools.

PHYSICAL DESCRIPTION

The ICE-5100/044 emulator consists of the following components (see Figure 1):

- Power supply
- AC and DC power cables
- Controller pod
- Serial Cable (host-specific)
- User probe assembly (consisting of the processor module and the user cable)
- Crystal power accessory (CPA)
- 40-pin target adaptor
- Clips assembly
- Software (includes the ICE-5100/044 emulator software, diagnostic software, and a tutorial)

The controller pod contains 64 KB of emulation memory, 254- by 24-bit frames of trace memory, and the control processor. In addition, the controller pod houses a BNC connector that can be used to connect up to 10 multi-ICE compatible emulators for synchronous starting and stopping of emulation.

The serial cable connects the host system to the controller pod. The serial cable supports a subset of the RS-232C signals.

The user probe assembly consists of a user cable and a processor module. The processor module houses the emulation processor and the interface logic. The target adaptor connects to the processor module and provides an electrical and mechanical interface to the target microcontroller socket.

The crystal power accessory (CPA) is a small, detachable board that connects to the controller pod and enables the ICE-5100/044 emulator to run in stand-alone mode. The target adaptor plugs into the socket on the CPA; the CPA then supplies clock and power to the user probe.

The clips assembly enables the user to trace external events. Eight bits of data are gathered on the rising edge of $\overline{PSEN}$ during opcode fetches. The clips information can be displayed using the CLIPS option with the PRINT command.

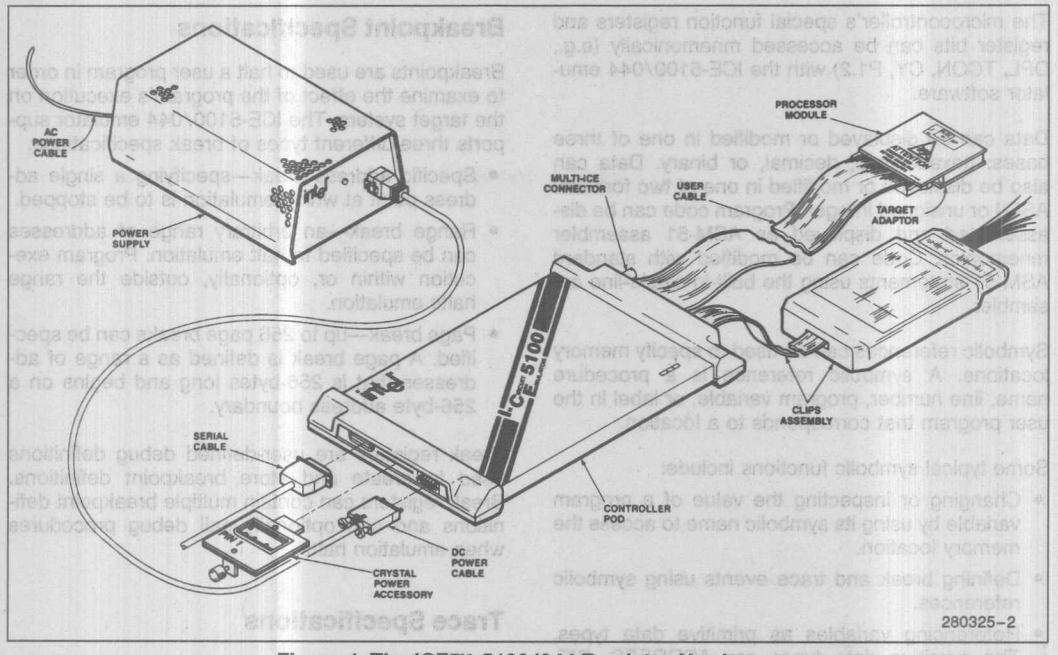


Figure 1. The ICE™-5100/044 Emulator Hardware

The ICE-5100-044 emulator software supports mnemonics, object file formats, and symbolic references generated by Intel's ASM-51 and PL/M-51 programming languages. Along with the ICE-5100/044 emulator software is a customer confidence test disk with diagnostic routines that check the operation of the hardware.

The on-line tutorial is written in the ICE-5100 command language. Thus, the user is able to interact with and use the ICE-5100/044 emulator while executing the tutorial.

A comprehensive set of documentation is provided with the ICE-5100/044 emulator.

ICE™-5100/044 EMULATOR FEATURES

The ICE-5100/044 emulator has been created to assist a product designer in developing, debugging and testing designs incorporating the RUPI-44 family of peripherals. The following sections detail some of the ICE-5100/044 emulator features.

Emulation

Emulation is the controlled execution of the user's software in the target hardware or in an artificial hardware environment that duplicates the microcontroller of the target system. Emulation is a transparent process that happens in real-time. The execution of the user software is facilitated with the ICE-5100/044 command language.

Memory Mapping

There is a 64 KB of memory that can be mapped to the CODE memory space in 4 KB blocks on 4 KB boundaries. By mapping memory to the ICE-5100/044 emulator, software development can proceed before the user hardware is available.

Memory Examination and Modification

The memory space for the 8044 microcontroller and its target hardware is fully accessible through the emulator. The ICE-5100/044 emulator refers to four physically distinct memory spaces, as follows:

- CODE—references program memory
- IDATA—references internal data memory
- RDATA—references special function register memory
- XDATA—references external data memory

ICE-5100/044 emulator commands that access memory use one of the special prefixes (e.g., CODE) to specify the memory space.

The microcontroller's special function registers and register bits can be accessed mnemonically (e.g., DPL, TCON, CY, P1.2) with the ICE-5100/044 emulator software.

Data can be displayed or modified in one of three bases: hexadecimal, decimal, or binary. Data can also be displayed or modified in one of two formats: ASCII or unsigned integer. Program code can be disassembled and displayed as ASM-51 assembler mnemonics. Code can be modified with standard ASM-51 statements using the built-in single-line assembler.

Symbolic references can be used to specify memory locations. A symbolic reference is a procedure name, line number, program variable, or label in the user program that corresponds to a location.

Some typical symbolic functions include:

- Changing or inspecting the value of a program variable by using its symbolic name to access the memory location.
- Defining break and trace events using symbolic references.
- Referencing variables as primitive data types. The primitive data types are ADDRESS, BIT, BOOLEAN, BYTE, CHAR (character), and WORD.

The ICE-5100/044 emulator maintains a virtual symbol table (VST) for program symbols. A maximum of 61 KB of host memory space is available for the VST. If the VST is larger than 61 KB, the excess is stored on available host system disk space and is paged in and out as needed. The size of the VST is limited only by the disk capacity of the host system.

Breakpoint Specifications

Breakpoints are used to halt a user program in order to examine the effect of the program's execution on the target system. The ICE-5100/044 emulator supports three different types of break specifications:

- Specific address break—specifying a single address point at which emulation is to be stopped.
- Range break—an arbitrary range of addresses can be specified to halt emulation. Program execution within or, optionally, outside the range halts emulation.
- Page break—up to 256 page breaks can be specified. A page break is defined as a range of addresses that is 256-bytes long and begins on a 256-byte address boundary.

Break registers are user-defined debug definitions used to create and store breakpoint definitions. Break registers can contain multiple breakpoint definitions and can optionally call debug procedures when emulation halts.

Trace Specifications

Tracing can be triggered using specifications similar to those used for breaking. Normally, the ICE-5100/044 emulator traces program activity while the user program is executing. With a trace specification, tracing can be triggered to occur only when specific conditions are met during execution. Up to 254 24-bit frames of trace information are collected in a buffer during emulation. Sixteen of the 24 bits trace instruction execution addresses, and 8 bits capture external events (CLIPS).

```
        /* Print newest four instructions in the buffer */
hlt>PRINT NEWEST 4
FRAME    ADDR      CODE      INSTRUCTIONS
(28)     300A      C02A      PUSH      2AH
(30      300C      2532      ADD       A, 32H
(32)     300E      F52A      MOV       2AH, A
(34)     3010      B53210    CJNE      A,32H, $+10H
hlt>
hlt>PRINT CLIPS OLDEST 2     /* Buffer display showing clips */
FRAME    ADDR      CODE      INSTRUCTIONS   CLIPS      (76543210)
(00)     007AH     0508      INC  INDX PTR              10101111
(01)     007CH     80E6      SJMP (#28)                 00100010
```

280325-3

Figure 2. Selected Trace Buffer Displays

The trace buffer display is similar to an ASM-51 program listing as shown in Figure 2. The PRINT command enables the user to selectively display the contents of the trace buffer. The user has the option of displaying the clips information as well as dissassembled instructions.

Procedures

Debugging procedures (PROCs) are a user-defined group of ICE-5100/044 commands that are executed as one command. PROCs enable the user to define several commands in a named block structure. The commands are executed by entering thename of the PROC. The PROC bodies are a simple DO . . . END construct.

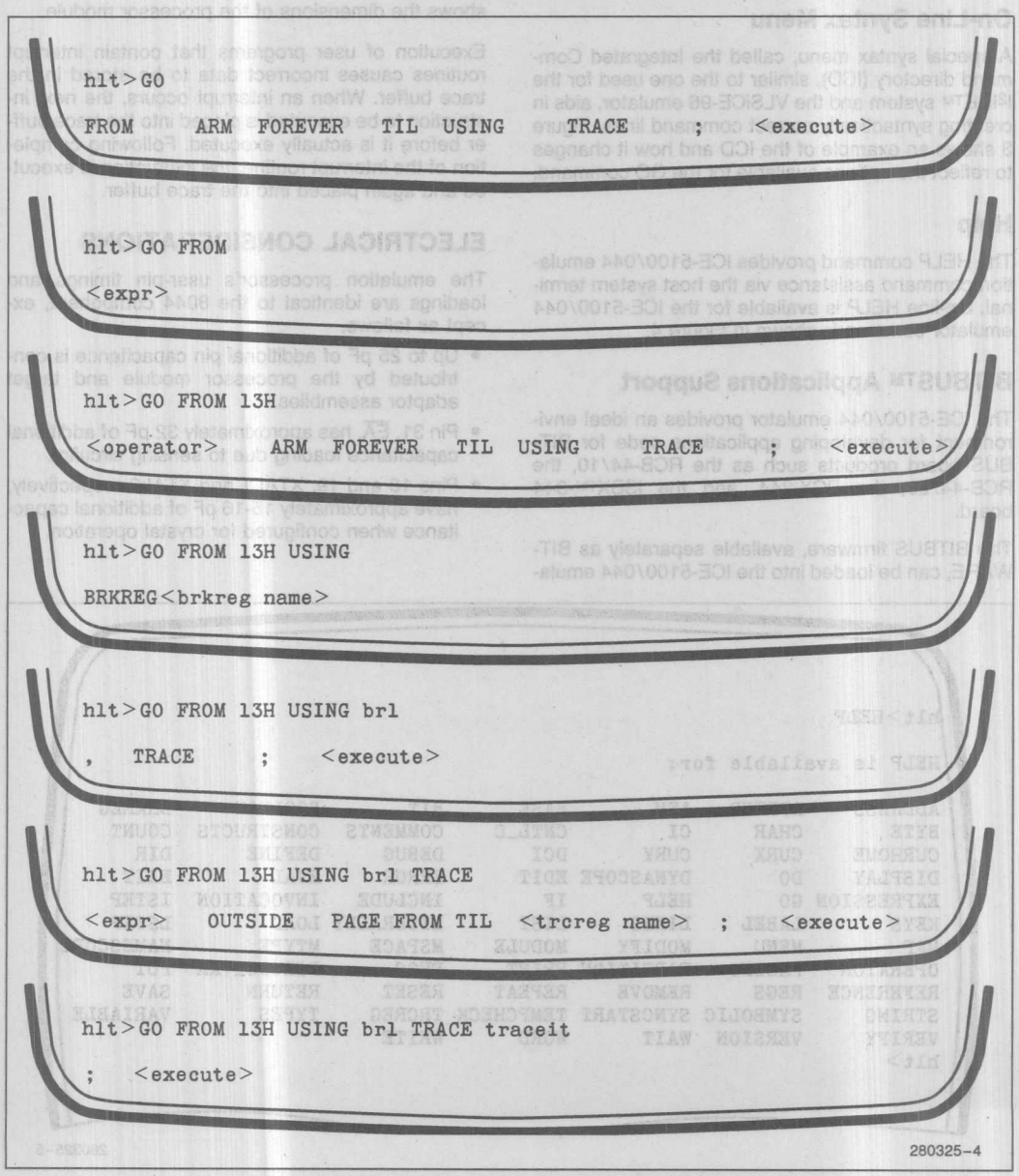


Figure 3. The Integrated Command Directory for the GO Command

PROCs can simulate missing hardware or software, set breakpoints, collect debug information, and execute high-level software patches. PROCs can be copied to text files on disk, then recalled for use in later test sessions. PROCs can also serve as program diagnostics, implementing ICE-5100/044 emulator commands or user-defined definitions for special purposes.

On-Line Syntax Menu

A special syntax menu, called the Integrated Command directory (ICD), similar to the one used for the I²ICE™ system and the VLSiCE-96 emulator, aids in creating syntactically correct command lines. Figure 3 shows an example of the ICD and how it changes to reflect the options available for the GO command.

Help

The HELP command provides ICE-5100/044 emulation command assistance via the host system terminal. On-line HELP is available for the ICE-5100/044 emulator commands shown in Figure 4.

BITBUS™ Applications Support

The ICE-5100/044 emulator provides an ideal environment for developing applications code for BITBUS board products such as the RCB-44/10, the RCB-44/20, the PCX-344, and the iSBX™-344 board.

The BITBUS firmware, available separately as BITWARE, can be loaded into the ICE-5100/044 emulator's memory along with the user's code to enable rapid debug of 8044 BITBUS applications code.

DESIGN CONSIDERATIONS

The height of the processor module and the target adaptor need to be considered for target systems. Allow at least 1½ inches (3.8 cm) of space to fit the processor module and target adaptor. Figure 5 shows the dimensions of the processor module.

Execution of user programs that contain interrupt routines causes incorrect data to be stored in the trace buffer. When an interrupt occurs, the next instruction to be executed is placed into the trace buffer before it is actually executed. Following completion of the interrupt routine, the instruction is executed and again placed into the trace buffer.

ELECTRICAL CONSIDERATIONS

The emulation processor's user-pin timings and loadings are identical to the 8044 component, except as follows.

- Up to 25 pF of additional pin capacitance is contributed by the processor module and target adaptor assemblies.
- Pin 31, $\overline{EA}$, has approximately 32 pF of additional capacitance loading due to sensing circuitry.
- Pins 18 and 19, XTAL1 and XTAL2 respectively, have approximately 15-16 pF of additional capacitance when configured for crystal operation.

```
hlt>HELP

HELP is available for:

ADDRESS     APPEND    ASM         BASE        BIT         BOOLEAN     BRKREG
BYTE        CHAR      CI          CNTL_C      COMMENTS    CONSTRUCTS  COUNT
CURHOME     CURX      CURY        DCI         DEBUG       DEFINE      DIR
DISPLAY     DO        DYNASCOPE   EDIT        ERROR       EVAL        EXIT
EXPRESSION  GO        HELP        IF          INCLUDE     INVOCATION  ISTEP
KEYS        LABEL     LINES       LIST        LITERALLY   LOAD        LSTEP
MAP         MENU      MODIFY      MODULE      MSPACE      MTYPE       NAMESCOPE
OPERATOR    PAGING    PARTITION   PRINT       PROC        PSEUDO_VAR  PUT
REFERENCE   REGS      REMOVE      REPEAT      RESET       RETURN      SAVE
STRING      SYMBOLIC  SYNCSTART   TEMPCHECK   TRCREG      TYPES       VARIABLE
VERIFY      VERSION   WAIT        WORD        WRITE
hlt>
```

280325-5

Figure 4. HELP Menu

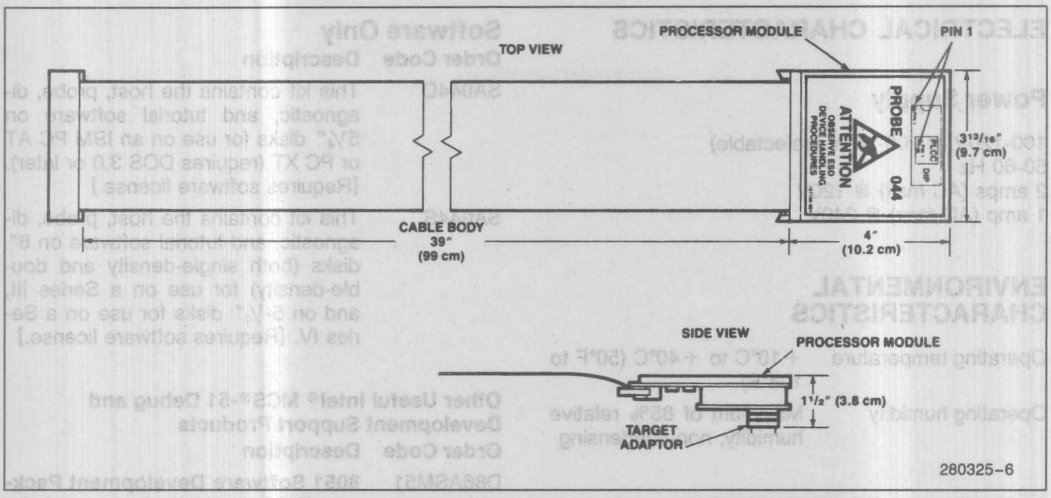


Figure 5. Processor Module Dimensions

HOST REQUIREMENTS

- IBM PC AT or PC XT (or PC DOS compatible) with 512 KB of available RAM and a hard disk running under the DOS 3.0 (or later) operating system.
- Intellec Series III/IV microcomputer development system running the ISIS or iNDX operating system respectively, with at least 512 KB of application memory resident.
- Disk drives—dual floppy or one hard disk and one floppy drive required.

ICE™-5100/044 EMULATOR SOFTWARE PACKAGE

- ICE-5100/044 emulator software
- ICE-5100/044 confidence tests
- ICE-5100 tutorial software

EMULATOR PERFORMANCE

Memory

Mappable full-speed emulation code memory	64 KB	Mappable to user or ICE-5100/044 emulator memory in 4 KB blocks on 4 KB boundaries.
Trace memory		254 x 24 bit frames
Virtual Symbol Table		A maximum of 61 KB of host memory space is available for the virtual symbol table (VST). The rest of the VST resides on disk and is paged in and out as needed.

PHYSICAL CHARACTERISTICS

Controller Pod

Width:	8-1/4"	(21 cm)
Height:	1-1/2"	(3.8 cm)
Depth:	13-1/2"	(34.3 cm)
Weight:	4 lbs	(1.85 kg)

User Cable

The user cable is 3 feet (approximately 1 m)

Processor Module

(With the target adaptor attached)

Width:	3-13/16"	(9.7 cm)
Height:	4"	(10.2 cm)
Depth:	1-1/2"	(3.8 cm)

Power Supply

Width:	7-5/8"	(18.1 cm)
Height:	4"	(10.06 cm)
Depth:	11"	(27.97 cm)
Weight	15 lbs	(6.1 kg)

Serial Cable

The serial cable is 12 feet (3.6 m).

ELECTRICAL CHARACTERISTICS

Power Supply

100-120V or 200-240V (selectable)
50-60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

Operating temperature	+10°C to +40°C (50°F to 104°F)
Operating humidity	Maximum of 85% relative humidity, non-condensing

ORDERING INFORMATION

Emulator Hardware and Software

Order Code	Description
I044KITAD	This kit contains the ICE-5100/044 user probe assembly, power supply and cables, serial cables, target adaptor, CPA, ICE-5100 controller pod, software, and documentation for use with an IBM PC AT or PC XT. The kit also includes the 8051 Software Development Package and the AEDIT text editor for use on DOS systems. [Requires software license.]
I044KITD	This kit is the same as the I044KITAD excluding the 8051 Software Development Package and the AEDIT text editor. [Requires software license.]
I044KITAS	This kit contains the ICE-5100/044 user probe assembly, power supply and cables, serial cables, target adaptor, CPA, ICE-5100 controller pod, software, and documentation for use with Intel hosts (Series III/IV). The kit also includes the 8051 Software Development Package and the AEDIT text editor for use on the Series III/IV. [Requires software license.]
I044KITS	This kit is the same as the I044KITAS excluding the 8051 Software Development Package and the AEDIT text editor. [Requires software license.]

Software Only

Order Code	Description
SA044D	This kit contains the host, probe, diagnostic, and tutorial software on 5¼" disks for use on an IBM PC AT or PC XT (requires DOS 3.0 or later). [Requires software license.]
SA044S	This kit contains the host, probe, diagnostic, and tutorial software on 8" disks (both single-density and double-density) for use on a Series III, and on 5-¼" disks for use on a Series IV. [Requires software license.]

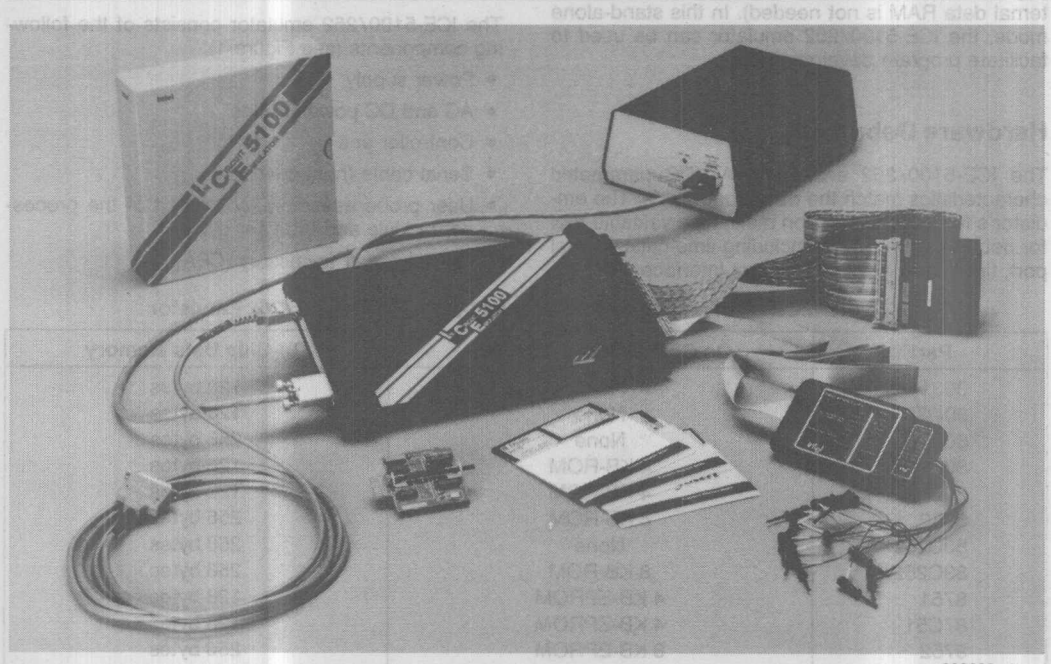
Other Useful Intel® MCS®-51 Debug and Development Support Products

Order Code	Description
D86ASM51	**8051 Software Development Package** (DOS version)—Consists of the ASM-51 macro assembler which gives symbolic access to 8051 hardware features; the RL51 linker and relocator program that links modules generated by ASM-51; CONV51 which enables software written for the MCS-48 family to be up-graded to run on the 8051, and the LIB51 Librarian which programmers can use to create and maintain libraries of software object modules. Use with the DOS operating system (version 3.0 or later).
D86PLM51	**PL/M-51 Software Package** (DOS version)—Consists of the PL/M-51 compiler which provides high-level programming language support; the LIB51 utility that creates and maintains libraries of software object modules, and the RL51 linker and relocator program that links modules generated by ASM-51 and PL/M-51 and locates the linked object modules to absolute memory locations. Use with the DOS operating system (version 3.0 or later).
I86ASM51	**8051 Software Development Package** (ISIS version)—Same as the D86ASM51 package except this one is for use with the Series III.
I86PLM51	**PL/M-51 Sofware Package**—Same as the D86PLM51 package except this one is for use with the Series III and Series IV.
D86EDINL	AEDIT text editor for use with the DOS operating system.

ICE™-5100/252 In-Circuit Emulator for the MCS®-51 Family of Microcontrollers

- **Precise, Full-Speed, Real-Time Emulation of Selected MCS-51 Microcontroller Components at Speeds Up to and Including 16 MHz**
- **64 KB of Mappable High-Speed Emulation Memory**
- **254 24-Bit Frames of Trace Memory (16 Bits Trace Program Execution Addresses and 8 Bits Trace External Events)**
- **Serial Link to the IBM* PC AT, PC XT (and DOS Compatibles), and the Intellec® Series III/IV**
- **ASM-51 and PL/M-51 Language Support**
- **Symbolic Debugging Enables Access to Memory Locations and Program Variables**
- **Four Address Breakpoints with In-Range, Out-Of-Range, and Page Breaks**
- **Equipped with the Integrated Command Directory (ICD™) that Includes:**
  - **— On-Line Help**
  - **— Syntax Guidance and Checking**
  - **— Dynamic Command-Entry**
  - **— Error Checking**
  - **— Command Recall**
- **On-Line Disassembler and Single-Line Assembler to Help with Code Patching**
- **Built-In CRT-Oriented Text Editor**

The ICE™-5100/252 in-circuit emulator is a high-level, interactive debugging system that is used to develop and test the hardware and software of a target system based on the MCS®-51 family of microcontrollers. The ICE-5100/252 emulator can be serially linked to an IBM PC AT or PC XT, or an Intellec Series III/IV. The emulator can communicate with the host system at standard baud rates up to 19.2K. The design of the emulator supports selected MCS-51 microcontroller components at speeds up to and including 16 MHz.



280200–1

*IBM is a registered trademark of International Business Machines Corporation.

January 1987
Order Number: 280200-002

PRODUCT OVERVIEW

The ICE-5100/252 emulator provides full emulation support for the MCS®-51 family members listed in Table 1.

The ICE-5100/252 emulator enables hardware and software development to proceed simultaneously. With the ICE-5100/252 emulator, prototype hardware can be added to the system as it is designed and software can be developed prior to the completion of the hardware prototype. Software and hardware integration can occur while the product is being developed.

The ICE-5100/252 emulator assists four stages of development:

- Software debugging
- Hardware debugging
- System integration
- System test

Software Debugging

The ICE-5100/252 emulator can be operated without being connected to the target system or before any of the user's hardware is available (provided external data RAM is not needed). In this stand-alone mode, the ICE-5100/252 emulator can be used to facilitate program development.

Hardware Debugging

The ICE-5100/252 emulator's AC/DC parametric characteristics match the microcontroller's. The emulator's full-speed operation makes it a valuable tool for debugging hardware, including time-critical serial port, timer, and external interrupt interfaces.

System Integration

Integration of software and hardware can begin when the emulator is plugged into the microcontroller socket of the prototype system hardware. Hardware can be added, modified, and tested immediately. As each section of the user's hardware is completed, it can be added to the prototype. Thus, the hardware and software can be system tested in real-time operation as each section becomes available.

System Test

When the prototype is complete, it is tested with the final version of the system software. The ICE-5100/252 emulator is then used for real-time emulation of the microcontroller to debug the system as a completed unit.

The final product verification test can be performed using the ROM or EPROM version of the microcontroller. Thus, the ICE-5100/252 emulator provides the ability to debug a prototype or production system at any stage in its development without introducing extraneous hardware or software test tools.

PHYSICAL DESCRIPTION

The ICE-5100/252 emulator consists of the following components (see Figure 1):

- Power supply
- AC and DC power cables
- Controller pod
- Serial cable (host-specific)
- User probe assembly (consisting of the processor module and the user cable)
- Crystal power accessory (CPA)

Table 1. MCS®-51 Family Support Offered by the ICE™-5100/252 Emulator

Part	On-Chip Program Memory	On-Chip Data Memory
8031	None	128 bytes
80C31	None	128 bytes
8032	None	256 bytes
8051	4 KB-ROM	128 bytes
80C51	4 KB-ROM	128 bytes
8052	8 KB-ROM	256 bytes
80C252	None	256 bytes
83C252	8 KB-ROM	256 bytes
8751	4 KB-EPROM	128 bytes
87C51	4 KB-EPROM	128 bytes
8752	8 KB-EPROM	256 bytes
87C252	8 KB-EPROM	256 bytes

- 40-pin DIP target adaptor
- Clips assembly
- Software (includes the ICE-5100/252 emulator software, diagnostic software, and tutorial)

The controller pod contains 64 KB of emulation memory, 254- by 24-bit frames of trace memory, and the control processor. In addition, the controller pod houses a BNC connector that can be used to connect up to 10 multi-ICE compatible emulators together for synchronous starting and stopping of emulation.

The serial cable connects the host system to the controller pod. The serial cable supports a subset of the RS-232C signals.

The user probe assembly consists of a user cable and a processor module. The processor module houses the emulation processor and the interface logic. The target adaptor connects to the processor module and provides an electrical and mechanical interface to the target microcontroller socket.

The crystal power accessory (CPA) is a small detachable board that connects to the controller pod and enables the ICE-5100/252 emulator to run in stand-alone mode. The target adaptor plugs into the socket on the CPA; the CPA then supplies clock and power to the user probe.

The clips assembly enables the user to trace external events. Eight bits of data are gathered on the rising edge of PSEN during opcode fetches. The clips information can be displayed using the CLIPS option with the PRINT command. Trace qualification input and output lines are also provided on the clips pod for connection to test equipment.

The ICE-5100/252 emulator software supports mnemonics, object file formats, and symbolic references generated by Intel's ASM-51 and PL/M-51 programming languages. Along with the ICE-5100/252 emulator software is a customer confidence test disk with diagnostic routines that check the operation of the hardware.

The on-line tutorial is written in the ICE-5100 command language. Thus, the user is able to interact with and use the ICE-5100/252 emulator while executing the tutorial.

A comprehensive set of documentation is included with the ICE-5100/252 emulator.

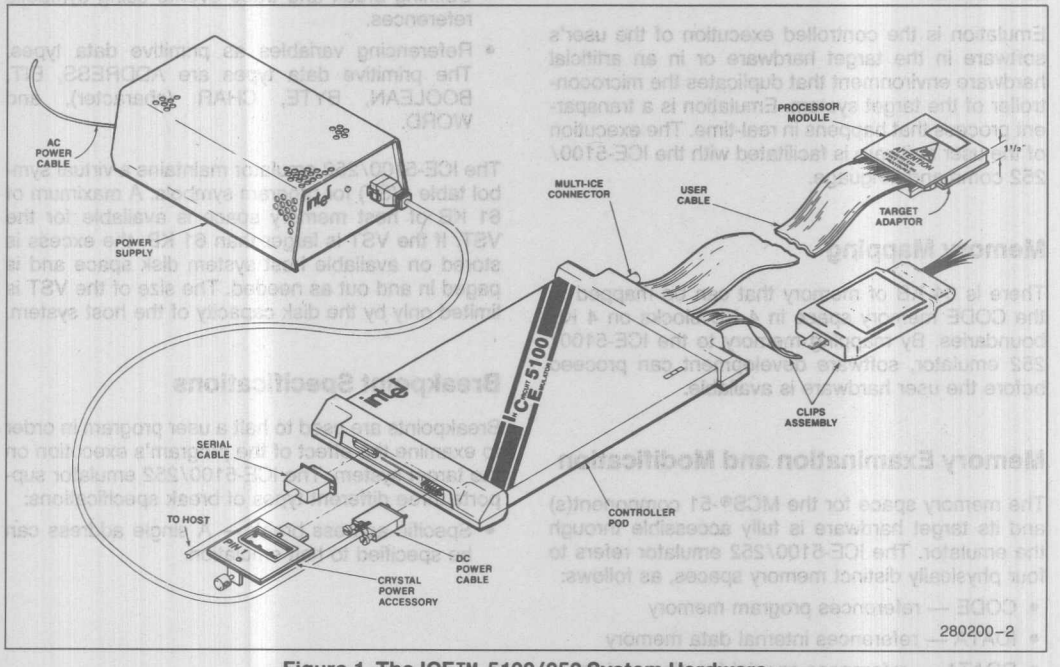


Figure 1. The ICE™-5100/252 System Hardware

ICE™-5100/252 EMULATOR FEATURES

The ICE-5100/252 emulator has been created to assist a product designer in developing, debugging, and testing designs incorporating the MCS®-51 family of microcontrollers. The following sections detail some of the ICE-5100/252 emulator features.

Processor Selection

The ICE-5100/252 emulator emulates the microcontrollers listed in Table 1. Selecting a processor type changes the following characteristics to match the microcontroller selected:

- Internal RAM size
- Internal ROM size
- Idle and power down mode enable
- Special function register symbolic map
- Memory map
- Latched or unlatched $\overline{EA}$
- Serial port framing and error detection

Emulation

Emulation is the controlled execution of the user's software in the target hardware or in an artificial hardware environment that duplicates the microcontroller of the target system. Emulation is a transparent process that happens in real-time. The execution of the user software is facilitated with the ICE-5100/252 command language.

Memory Mapping

There is 64 KB of memory that can be mapped to the CODE memory space in 4 KB blocks on 4 KB boundaries. By mapping memory to the ICE-5100/252 emulator, software development can proceed before the user hardware is available.

Memory Examination and Modification

The memory space for the MCS®-51 component(s) and its target hardware is fully accessible through the emulator. The ICE-5100/252 emulator refers to four physically distinct memory spaces, as follows:

- CODE — references program memory
- IDATA — references internal data memory
- RDATA — references special function register memory
- XDATA — references external data memory

ICE-5100/252 emulator commands that access memory use one of the special prefixes (e.g., CODE) to specify the memory space.

The microcontroller's special function registers and register bits can be accessed mnemonically (e.g., DPL, TCON, CY) with the ICE-5100/252 emulator software.

Data can be displayed or modified in one of three bases: hexadecimal, decimal, and binary. Data can also be displayed or modified in one of two formats: ASCII and unsigned integer. Program code can be disassembled and displayed as ASM-51 assembler mnemonics. Code can be modified with standard ASM-51 statements using the built-in single-line assembler.

Symbolic references can be used to specify memory locations. A symbolic reference is a procedure name, line number, program variable, or label in the user program that corresponds to a location.

Some typical symbolic functions include:

- Changing or inspecting the value of a program variable by using the symbolic name to access the memory location.
- Defining break and trace events using symbolic references.
- Referencing variables as primitive data types. The primitive data types are ADDRESS, BIT, BOOLEAN, BYTE, CHAR (character), and WORD.

The ICE-5100/252 emulator maintains a virtual symbol table (VST) for program symbols. A maximum of 61 KB of host memory space is available for the VST. If the VST is larger than 61 KB, the excess is stored on available host system disk space and is paged in and out as needed. The size of the VST is limited only by the disk capacity of the host system.

Breakpoint Specifications

Breakpoints are used to halt a user program in order to examine the effect of the program's execution on the target system. The ICE-5100/252 emulator supports three different types of break specifications:

- Specific address break — A single address can be specified to halt emulation.

- Range break — An arbitrary range of addresses can be specified to halt emulation. Program execution within or, optionally, outside the range halts emulation.
- Page break — Up to 256 page breaks can be specified. A page break is defined as a range of addresses that is 256-bytes long and begins on a 256-byte address boundary.

Break registers are user-defined debug definitions used to create and store breakpoint definitions. Break registers can contain multiple breakpoint definitions and can optionally call debug procedures when emulation halts.

Trace Specifications

Tracing can be triggered using specifications similar to those used for breaking. Normally, the ICE-5100/252 emulator traces program activity while the user program is executing. With a trace specification, tracing can be triggered to occur only when specific conditions are met during execution. Up to 254 24-bit frames of trace information are collected in the buffer during emulation. Sixteen of the 24 bits trace instruction execution addresses, and 8 bits capture external events (CLIPS).

The trace buffer display is similar to an ASM-51 program listing shown in Figure 2. The PRINT command enables the user to selectively display the contents of the trace buffer. The user has the option of displaying the clips information as well as disassembled instructions.

Procedures

Debugging procedures (PROCs) are a user-named group of ICE-5100/252 emulator commands that are executed as one command. PROCs enable the user to define several commands in a named block structure. The commands are executed by entering the name of the PROC. The PROC bodies are a simple DO...END construct.

PROCs can simulate missing hardware or software, collect debug information, and execute high-level software patches. PROCs can be copied to text files on disk, then recalled for use in later test sessions. PROCs can also serve as program diagnostics, implementing ICE-5100/252 emulator commands or user-defined definitions for special purposes. PROCs can also be used to set breakpoints.

On-Line Syntax Menu

A special menu, called the Integrated Command Directory (ICD), similar to the one used for the I²ICE™ system and the VLSiCE-96 emulator, aids in creating syntactically correct command lines. Figure 3 shows an example of the ICD and how it changes to reflect the options available for the GO command.

Help

The HELP command provides ICE-5100/252 emulation command assistance via the host system terminal. On-line HELP is available for the ICE-5100/252 emulator commands shown in Figure 4.

```
hlt>PRINT NEWEST 4      /* Print newest four instructions in
                           buffer */
FRAME     ADDRESS  CODE    INSTRUCTION
(028)     300A     C02A    PUSH  2AH
(030)     300C     2532    ADD   A,32H
(032)     300E     F52A    MOV   2AH,A
(034)     3010     B53210  CJNE  A,32H,$+10H
hlt>
hlt>PRINT CLIPS OLDEST 2  /* Buffer display showing clips */
FRAME     ADDRESS  CODE    INSTRUCTION    CLIPS (76543210)
(000)     300A     C02A    PUSH  2AH        01110011
(001)     300C     2532    ADD   A,32H      11110101
hlt>
```

280200-3

Figure 2. Selected Trace Buffer Displays

Design Considerations

The height of the processor module and the target adaptor need to be considered for target systems. Allow at least 1-½ inches (3.8 cm) of space to fit the processor module and target adaptor. Figure 5 shows the dimensions of the processor module.

```
hlt> GO
FROM  ARM  FOREVER   TIL  USING  TRACE  ;  <execute>

hlt> GO FROM
<expr>

hlt> GO FROM 13H
<operator> ARM FOREVER  TIL USING  TRACE ; <execute>

hlt> GO FROM 13H USING
BRKREG <brkreg name>

hlt> GO FROM 13H USING brk
; TRACE  ;  <execute>

hlt> GO FROM 13H USING brk TRACE traceit
;  <execute>
```

280200-4

Figure 3. The Integrated Command Directory for the GO Command

```
hlt>HELP

HELP is available for:

ADDRESS    APPEND     ASM        BASE       BIT        BOOLEAN
BRKREG     BYTE       CHAR       CI         CNTL_C     COMMENTS
CONSTRUCTS COUNT      CPU        CURHOME    CURX       CURY
DCI        DEBUG      DEFINE     DIR        DISPLAY    DO
DYNASCOPE  EDIT       ERROR      EVAL       EXIT       EXPRESSION
GO         HELP       IF         INCLUDE    INVOCATION ISTEP
KEYS       LABEL      LINES      LIST       LITERALLY  LOAD
LSTEP      MAP        MENU       MODIFY     MODULE     MSPACE
MTYPE      NAMESCOPE  OPERATOR   PAGING     PARTITION  PRINT
PROC       PSEUDO_VAR PUT        REFERENCE  REGS       REMOVE
REPEAT     RESET      RETURN     SAVE       STRING     SYMBOLIC
SYNCSTART  TEMP.CHECK TRCREG     TYPES      VARIABLE   VERIFY
VERSION    WAIT       WORD       WRITE
hlt>
```

280200-5

Figure 4. HELP Menu

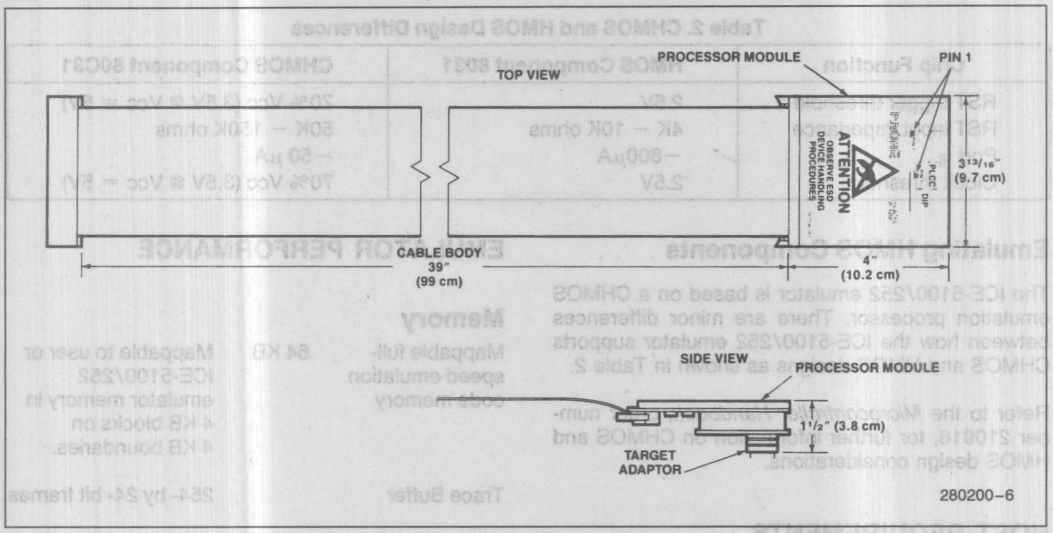


Figure 5. Processor Module Dimensions

ELECTRICAL CONSIDERATIONS

The emulation processor's user-pin timings and loadings are identical to the 80C252 component except as follows.

Maximum Operating ICC and Idle ICC (ma)*

	Maximum Operating ICC (ma)*			Maximum Idle ICC (ma)*		
V_{CC}	4V	5V	6V	4V	5V	6V
	Frequency					
0.5 MHz	0.87	1.62	3.0	0.58	1.21	2.5
3.5 MHz	4.8	6.82	9.76	2.2	4.97	6.33
8.0 MHz	10.5	15.0	20.5	6.0	8.98	11.76
12.0 MHz	15.2	22.2	30.2	9.2	13.34	17.46
16.0 MHz	19.4	28.6	38.7	11.8	17.4	23.4

* ICC is measured with all output pins disconnected.
XTAL1 driven with TCLCH, TCHCL = 10ns, $V_{il} = V_{ss} + .5V$, $V_{ih} = Vcc - .5V$. XTAL2 not connected.
For maximum operating ICC
$\overline{EA}$ = RST = Port0 = Vcc.
For maximum idle ICC
* $\overline{EA}$ = Port0 = Vcc, RST = Vcc, internal clock to PCA gated off.

- Up to 25 pf of additional pin capacitance is contributed by the processor module and target adaptor assemblies.
- Pin 31, $\overline{EA}$, has approximately 32 pf of additional capacitance loading due to sensing circuitry.
- Pins 18 and 19, XTAL1 and XTAL2, respectively, have approximately 15 to 16 pf of additional capacitance when configured for crystal operation.

Table 2. CHMOS and HMOS Design Differences

Chip Function	HMOS Component 8031	CHMOS Component 80C31
RST trigger threshold	2.5V	70% Vcc (3.5V @ Vcc = 5V)
RST input impedance	4K – 10K ohms	50K – 150K ohms
Port I_{IL}	$-800\mu A$	$-50\ \mu A$
Clock threshold	2.5V	70% Vcc (3.5V @ Vcc = 5V)

Emulating HMOS Components

The ICE-5100/252 emulator is based on a CHMOS emulation processor. There are minor differences between how the ICE-5100/252 emulator supports CHMOS and HMOS designs as shown in Table 2.

Refer to the *Microcontroller Handbook*, order number 210918, for further information on CHMOS and HMOS design considerations.

HOST REQUIREMENTS

- IBM PC AT or PC XT (or PC-DOS compatible) with 512 KB of RAM and a hard disk running under the DOS 3.0 (or later) operating system.
- Intellec Series III/IV Microcomputer Development System running under the ISIS or iNDX operating system respectively, with at least 512 KB of application memory resident.

  Disk drives — Dual floppy or one hard disk and one floppy drive required.

ICE™-5100/252 SYSTEM SOFTWARE PACKAGE

- ICE-5100/252 emulator software
- ICE-5100/252 confidence tests
- ICE-5100/252 tutorial software

EMULATOR PERFORMANCE

Memory

Mappable full-speed emulation code memory	64 KB	Mappable to user or ICE-5100/252 emulator memory in 4 KB blocks on 4 KB boundaries.
Trace Buffer		254- by 24- bit frames
Virtual Symbol Table		A maximum of 61 KB of host memory space is available for the Virtual Symbol Table (VST). The rest of the VST resides on disk and is paged in and out as needed.

PHYSICAL CHARACTERISTICS

Controller Pod

Width	8¼" (21 cm)
Height	1½" (3.8 cm)
Depth	13½" (34.3 cm)
Weight	4 lbs (1.85 kg)

User Cable

The user cable is 3 feet (approximately 1 m).

Processor Module

(with the target adaptor attached)

Width	3 13/16" (9.7 cm)
Length	1½" (3.8 cm)
Height	1½" (3.8 cm)

Power Supply

Width	7⅝" (18.1 cm)
Height	4" (10.06 cm)
Depth	11" (27.97 cm)
Weight	15 lbs (6.1 kg)

Serial Cable

The serial cable is 12 feet (3.6 m).

ELECTRICAL CHARACTERISTICS

Power Supply

100 - 120V or 200 - 240V (selectable)
50 - 60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

Operating temperature	+10° C to +40°C (50°F to 104°F)
Operating humidity	Maximum of 85% relative humidity, non-condensing

ORDERING INFORMATION

Emulator Hardware and Software

Order Code	Description
pI252KITAD	This kit contains: ICE-5100/252 user probe assembly, power supply and cables, serial cables, target adaptor, CPA, ICE-5100 controller pod, software, and documentation for use with an IBM PC AT or PC XT. The kit also includes the 8051 Software Development Package and the AEDIT text editor for use on DOS systems. [Requires software license.]
pI252KITD	This kit is the same as the pI252KITAD kit excluding the 8051 Software Development Package and the AEDIT text editor. [Requires software license.]
pI252KITAS	This kit contains the ICE-5100/252 user probe assembly, power supply and cables, serial cables, target adaptor, CPA, ICE-5100 controller pod, software, and documentation for use with Intel hosts (Series III, IV). The kit also includes the 8051 Software Development Package and the AEDIT text editor for use on Series III/IV. [Requires software license.]
pI252KITS	This kit is the same as the pI252KITAS kit excluding the 8051 Software Development Package and the AEDIT text editor. [Requires software license.]

Software Only

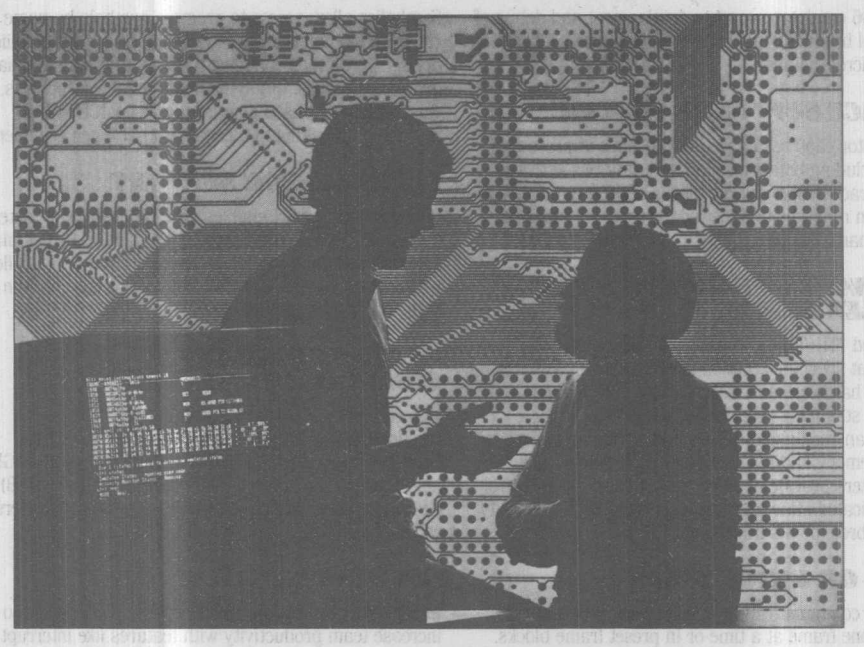
Order Code	Description
pSA252D	This kit contains the host, probe, diagnostic and tutorial software on 5¼" disks for use on an IBM PC AT or PC XT (requires DOS 3.0 or later). [Requires software license.]
pSA252S	This kit contains the host, probe, diagnostic and tutorial software on 8" disks (both single-density and double-density) for use on a Series III, and on 5¼" disks for use on a Series IV. [Requires software license.]

Other Useful Intel Debug and Development Support Products

Order Code	Description
pD86ASM51	8051 Software Development Package (DOS version) — Consists of the ASM-51 macro assembler which gives symbolic access to 8051 hardware features; the RL51 linker and relocator program that links modules generated by ASM-51; CONV51 which enables software written for the MCS-48 family to be up-graded to run on the 8051, and the LIB51 Librarian which programmers can use to create and maintain libraries of software object modules. Use with the DOS operating system (version 3.0 or later).

pD86PLM51	PL/M-51 Software Package (DOS version) — Consists of the PL/M-51 compiler which provides high-level programming language support; the LIB51 utility that creates and maintains libraries of software object modules, and the RL51 linker and relocator program that links modules generated by ASM-51 and PL/M-51 and locates the linked object modules to absolute memory locations. Use the DOS operating system (version 3.0 or later).
pI86ASM51	8051 Software Development Package (ISIS version) — Same as the pD86ASM51 package except this one is for use with the Series III.
pI86PLM51	PL/M-51 Software Package — Same as the pD86PLM51 package except this one is for use with the Series III and Series IV.
pD86EDIEU	AEDIT text editor for use with the DOS operating system.

ICE™-186 IN-CIRCUIT EMULATOR



HIGH PERFORMANCE REAL-TIME EMULATION

Intel's ICE-186 emulator delivers real-time emulation for the 80C186 microprocessor at speeds up to 12.5 MHz. The in-circuit emulator is a versatile and efficient tool for developing, debugging and testing products designed with the Intel 80C186 microprocessor. The ICE-186 emulator provides real time, full speed emulation in a users system. Popular features such as symbolic debug, 2K bytes trace memory, and single-step program execution are standard on the ICE-186 emulator. Intel provides a complete development environment using assembler (ASM86) as well as high-level languages such as Intel's C86, PL/M86 or Fortran 86 to accelerate development schedules.

The ICE-186 emulator supports a subset of the 80C186 features at 12.5 MHz and at the TTL level characteristics of the component. The emulator is hosted on IBM's Personal Computer AT, already available as a standard development solution in most of today's engineering environments. The ICE-186 emulator operates in prototype or standalone mode, allowing software development and debug before a prototype system is available. The ICE-186 emulator is ideally suited for developing real-time applications such as industrial automation, computer peripherals, communications, office automation, or other applications requiring the full power of the 12.5 MHz 80C186 microprocessor.

ICE™-186 FEATURES

- Full 12.5 MHz Emulation Speed
- 2K Bytes Deep Trace Memory
- Two-Level Breakpoints with Occurrence Counters
- Single-Step Capability
- 128K Bytes Zero Wait-State Mapped Memory
- Supports DRAM Refresh
- High-Level Language Support
- Symbolic Debug
- Coprocessor Support
- RS-232-C and GPIB Communication Links
- Crystal Power Accessory
- Interface for Intel Performance Analysis Tool (iPAT)*
- Interface for Optional General Purpose Logic Analyzer
- Tutorial Software
- Complete Intel Service and Support

intel

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

© Intel Corporation 1987

AUGUST, 1987
Order Number: 280726-001

HIGHEST EMULATION SPEED AVAILABLE TODAY

The ICE-186 emulator supports development and debug of time-critical hardware and software using Intel's 12.5 MHz 80C186 microprocessor.

RETRACE SOFTWARE TRACKS

This emulator captures up to 2,048 frames of processor activity, including both execution and data bus activity. With this trace memory, large blocks of program code can be traced in real time and viewed for program flow and behavior characteristics.

HARDWARE BREAKPOINTS FOR COMPLEX DEBUG

User-defined "TIL-THEN" breakpoint statements stop emulation at specific execution addresses or bus events. During the hardware and software integration phase, breakpoint statements can be defined as execution addresses and/or bus addresses and/or bus access types such as memory and I/O reads or writes. Additionally, event counters provide another level of breakpoint control for sophisticated state machine constructs used to specify emulation breakpoints/tracepoints.

SMALL OR LARGE STEPS

A stepping command can be used to view program execution one frame at a time or in preset frame blocks. When used in conjunction with symbolic debug, code execution can be monitored quickly and precisely.

DEBUG CODE WITHOUT A PROTOTYPE

Even before prototype hardware is available, the ICE-186 emulator working in conjunction with the Crystal Power Accessory (CPA) creates a "virtual" application environment. 128K bytes of zero wait-state memory is available for mapped memory and I/O resource addressing in 4K increments. The CPA provides emulator diagnostics as well as the ability to use the emulator without a prototype.

DON'T LOSE MEMORY

The ICE-186 emulator continues DRAM refresh signals even when emulation has been halted, thus ensuring DRAM memory will not be lost. During interrogation mode the ICE-186 emulator will keep the timers functioning and correctly respond to interrupts in real-time.

HIGH LEVEL LANGUAGE SUPPORT OPTIMIZED FOR INTEL TOOLS

The ICE-186 supports emulation for programs written in Intel's ASM86 or any of Intel's high-level languages:

PL/M-86	Fortran-86
Pascal-86	C-86

These languages are optimized for Intel component architectures to deliver a tightly integrated, high performance development environment.

USER-FRIENDLY SYMBOLICS AID IN DEBUG

Symbolics allow access to program symbols by name rather than cumbersome physical addresses. Symbolic debug speeds the debugging process by reducing reliance on memory maps. In a dynamic development process, user variables can be used as parameters for ICE-186 commands resulting in a consistent debug environment.

COPROCESSOR SUPPORT

Coprocessor support enables applications to run faster due to off loading of the main CPU. The ICE-186 emulator supports alternate coprocessors such as LAN controllers and graphic engines, however it does not have built in support for the 8087 coprocessor.

MULTIPLE HIGH-SPEED COMMUNICATION LINKS

Two communication links are available for use in conjunction with the host IBM PC AT. The ICE-186 emulator uses either serial (RS-232-C) or a parallel (GPIB) link. A user supplied National Instruments (IEEE-488) GPIB communication board provides parallel transfers at rates up to 300K bytes per second.

SOFTWARE ANALYSIS (iPAT)

Intel's Performance Analysis Tool (iPAT) is designed to increase team productivity with features like interrupt latency measurement, code coverage analysis and software module performance analysis. These features enable the user to design reliable, high performance embedded control products. The ICE-186 emulator has an external 60 pin connector for iPAT.

BUILT-IN SUPPORT FOR LOGIC ANALYSIS

General-purpose logic analyzers can be used in conjunction with the ICE-186 to provide detailed timing of specific events. The ICE-186 emulator provides an external sync signal for triggering logic analysis, making complex trigger sequence programming easy. An additional 60 pin connector is included for the logic analyzer.

WORLDWIDE SERVICE AND SUPPORT

The ICE-186 emulator is supported by Intel's worldwide service and support organization. Total hardware and software support is available including a hotline number when the need is there.



SPECIFICATIONS

PERSONAL COMPUTER REQUIREMENTS

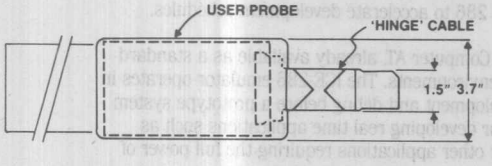
The ICE-186 emulator is hosted on an IBM PC AT. The emulator has been tested and evaluated on an IBM PC AT. The PC AT must meet the following minimum requirements:

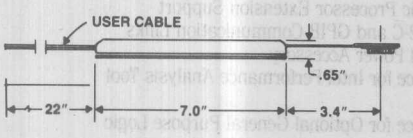
- 640K Bytes of Memory
- Intel Above Board with at Least 1M Byte of Expansion Memory
- One 360K Bytes or One 1.2M Bytes floppy Disk Drive
- One 20M Bytes Fixed-Disk Drive
- PC DOS 3.2 or Later
- A serial Port (COM1 or COM2) Supporting Minimally at 9600 Baud Data Transfers, or a National Instruments GPIB-PC2A board.
- IBM PC AT BIOS

PHYSICAL DESCRIPTION AND CHARACTERISTICS

The ICE-186 Emulator consists of the following components:

Unit	Width		Height		Length	
	Inches	Cm.	Inches	Cm.	Inches	Cm.
Emulator Control Unit	10.40	26.40	1.70	4.30	20.70	52.60
Power Supply	2.80	7.10	4.15	10.70	11.00	27.90
User Probe	3.70	9.40	.65	1.60	7.00	17.80
User Cable/ Plcc					22.00	55.90
Hinge Cable					3.40	8.60
Crystal Power Accessory	4.30	10.90	.60	1.50	6.70	17.00
CPA Power Cable					9.00	22.90





ELECTRICAL CONSIDERATIONS

Icc 1050mA

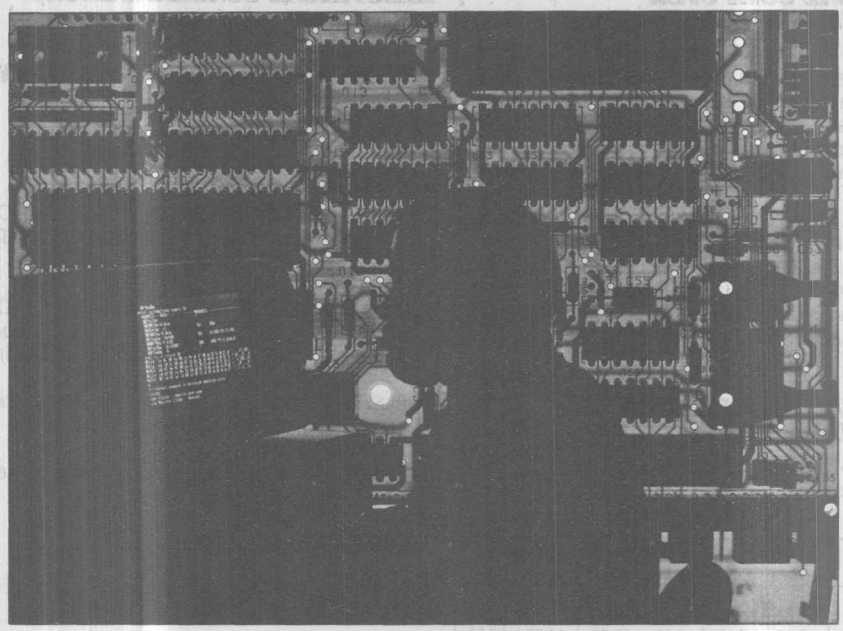
ENVIRONMENTAL SPECIFICATIONS

Operating Temperature 10°C-40°C Ambient
Storage Temperature −40°C-70°C

ORDERING INFORMATION

ICE186	ICE-186 NMOS System including ICE software (Requires DOS 3.XX PC AT with Above Board)
ICE186PAT	ICE-186 NMOS System including ICE S/W packages and the iPAT system (Requires DOS 3.XX PC AT with Above Board)
D86ASM86NL	86 macro assembler 86 builder/binder/ mapper utilities for DOS 3.XX.
D86C86NL	86 C compiler and run time libraries for DOS 3.XX.
D86PLM86NL	86 PL/M compiler for DOS 3.XX.
D86FOR86NL	86 Fortran compiler for DOS 3.XX.

ICE™-286 IN-CIRCUIT EMULATOR



HIGH PERFORMANCE REAL-TIME EMULATION

Intel's ICE-286 emulator delivers real-time emulation for the 80286 microprocessor at speeds up to 12.5 MHz. The in-circuit emulator is a versatile and efficient tool for developing, debugging and testing products designed with the Intel 80286 microprocessor. The ICE-286 emulator provides real time, full speed emulation in a users system. Popular features such as symbolic debug, 2K bytes trace memory, and single-step program execution are standard on the ICE-286 emulator. Intel provides a complete development environment using assembler (ASM-286) as well as high-level languages such as Intel's C286, PL/M-286 or Fortran 286 to accelerate development schedules.

Intel's ICE-286 emulator is hosted on IBM's Personal Computer AT, already available as a standard development solution in most of today's engineering environments. The ICE-286 emulator operates in prototype or standalone mode allowing software development and debug before a prototype system is available. The ICE-286 emulator is ideally suited for developing real time applications such as process control, machine control, communications, or other applications requiring the full power of the 12.5 MHz 80286 microprocessor.

ICE-286 FEATURES

- Full 12.5 MHz Emulation Speed
- 2K Bytes Deep Trace Memory
- Two-Level Breakpoints with Occurrence Counters
- Single-Step Capability
- 128K Bytes Zero Wait-State Mapped Memory
- Support For Protected and Real Modes
- High-Level Language Support
- Symbolic Debug
- Numeric Processor Extension Support
- RS-232-C and GPIB Communication Links
- Crystal Power Accessory
- Interface for Intel Performance Analysis Tool (iPAT)
- Interface for Optional General Purpose Logic Analyzer
- Tutorial Software
- Complete Intel Service and Support

intel

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

© Intel Corporation 1987

AUGUST, 1987
Order Number: 280728-001

HIGHEST EMULATION SPEED AVAILABLE TODAY

The ICE-286 emulator supports development and debug of time-critical hardware and software using Intel's 12.5 MHz 80286 microprocessor.

RETRACE SOFTWARE TRACKS

This emulator captures up to 2048 frames of processor activity, including both execution and data bus activity. With this trace memory, large blocks of program code can be traced in real time and viewed for program flow and behavior characteristics.

HARDWARE BREAKPOINTS FOR COMPLEX DEBUG

User-defined "TIL-THEN" breakpoint statements stop emulation at specific execution addresses or bus events. During the hardware and software integration phase, breakpoint statements can be defined as execution addresses and/or bus addresses and/or bus access types, such as memory and I/O reads or writes. Additionally, event counters provide another level of breakpoint control for sophisticated state machine constructs used to specify emulation breakpoints/tracepoints.

SMALL OR LARGE STEPS

A stepping command can be used to view program execution one frame at a time or in preset frame blocks. When used in conjunction with symbolic debug, code execution can be monitored quickly and precisely.

DEBUG CODE WITHOUT A PROTOTYPE

Even before prototype hardware is available, the ICE-286 emulator working in conjunction with the Crystal Power Accessory (CPA) creates a "virtual" application environment. 128K bytes of zero wait-state memory is available for mapped memory and I/O resource addressing in 4K increments. The CPA provides emulator diagnostics as well as the ability to use the emulator without a prototype.

PROTECTED AND REAL MODES

The ICE-286 emulator has full access to all protected-mode registers and permits modification of register contents. Protected mode of execution if beneficial for secure, multitasking applications.

HIGH-LEVEL LANGUAGE SUPPORT OPTIMIZED FOR INTEL TOOLS

The ICE-286 supports emulation for programs written in Intel's ASM 286 and ASM 86 or any of the Intel high-level languages:

PL/M-286/86	Fortran-286/86
Pascal-286/86	C-286/86

These languages are optimized for Intel component architectures to deliver a tightly integrated, high performance development environment.

USER-FRIENDLY SYMBOLICS AID IN DEBUG

Symbolics allow access to program symbols by name rather than cumbersome physical addresses. Symbolic debug speeds the debugging process by reducing reliance on memory maps. In a dynamic development process, user variables can be used as parameters for ICE-286 commands resulting in a consistent debug environment.

80287 NUMERICS SUPPORT

The ICE-286 emulator provides emulation support for the 80287 numerics processor. 80287 registers can be displayed and modified allowing full debug support for numerics.

MULTIPLE HIGH-SPEED COMMUNICATION LINKS

Two communication links are available for use in conjunction with the host IBM PC AT. The ICE-286 emulator uses either serial (RS-232-C) or a parallel (GPIB) link. A user supplied National Instruments (IEEE-488) GPIB communication board provides parallel transfers at rates up to 300K bytes per second.

SOFTWARE ANALYSIS (iPAT)

Intel's Performance Analysis Tool (iPAT) is designed to increase team productivity with features like interrupt latency measurement, code coverage analysis and software module performance analysis. These features enable the user to design reliable, high performance embedded control products. The ICE-286 emulator has an external 60 pin connector for iPAT.

BUILT-IN SUPPORT FOR LOGIC ANALYSIS

General-purpose logic analyzers can be used in conjunction with the ICE-286 to provide detailed timing of specific events. The ICE-286 emulator provides an external sync signal for triggering logic analysis, making complex trigger sequence programming easy. An additional 60 pin connector is included for the logic analyzer.

WORLDWIDE SERVICE AND SUPPORT

The ICE-286 emulator is supported by Intel's worldwide service and support organization. Total hardware and software support is available including a hotline number when the need is there.



SPECIFICATIONS

PERSONAL COMPUTER REQUIREMENTS

The ICE-286 emulator is hosted on an IBM PC AT. The emulator has been tested and evaluated on an IBM PC AT. The PC AT must meet the following minimum requirements:

- 640K Bytes of Memory
- Intel Above Board with at Least 1M Byte of Expansion Memory
- One 360K Bytes or One 1.2M Bytes Floppy Disk Drive
- One 20M Bytes Fixed-Disk Drive
- PC-DOS 3.2 or Later
- A Serial Port (COM1 or COM2) Supporting Minimally at 9600 Baud Data Transfers, or a National Instruments GPIB-PC2A Board.
- IBM PC AT BIOS

ELECTRICAL CONSIDERATIONS

Icc 1050mA

ENVIRONMENTAL SPECIFICATIONS

Temperature 10°C to 40°C Ambient
Storage Temperature −40°C to 70°C

TIMING/DC CONSIDERATIONS

ICE™-286 USER PIN DIFFERENCES

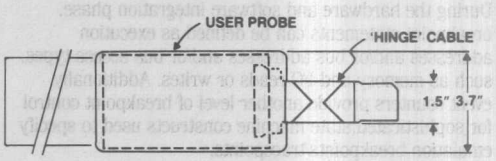
	PARAMETER	COMPONENT SPEC. Min	COMPONENT SPEC. Max	ICE-286 SPEC. Min	ICE-286 SPEC. Max
2	System (CLK) low time	11	237	14	236
3	System (CLK) high time	13	239	14	236
5	Asynch input hold time	15		17	
12a	Status/peak # active delay	3	18	3	20
12b	Status/peak # inactive delay	3	20	3	22
13	Address valid delay	1	32	1	34
14	Write data valid delay	0	30	0	33
15	Address/status/data float	0	32	0	34
16	HLDA valid delay	0	25	0	27

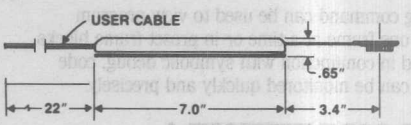
Consult User Guide for additional specifications.

PHYSICAL DESCRIPTION AND CHARACTERISTICS

The ICE-286 Emulator consists of the following components:

Unit	Width Inches	Width Cm.	Height Inches	Height Cm.	Length Inches	Length Cm.
Emulator Control Unit	10.40	26.40	1.70	4.30	20.70	52.60
Power Supply	2.80	7.10	4.15	10.70	11.00	27.90
User Probe	3.70	9.40	.65	1.60	7.00	17.80
User Cable/ Plcc					22.00	55.90
Hinge Cable					3.40	8.60
Crystal Power Accessory	4.30	10.90	.60	1.50	6.70	17.00
CPA Power Cable					9.00	22.90

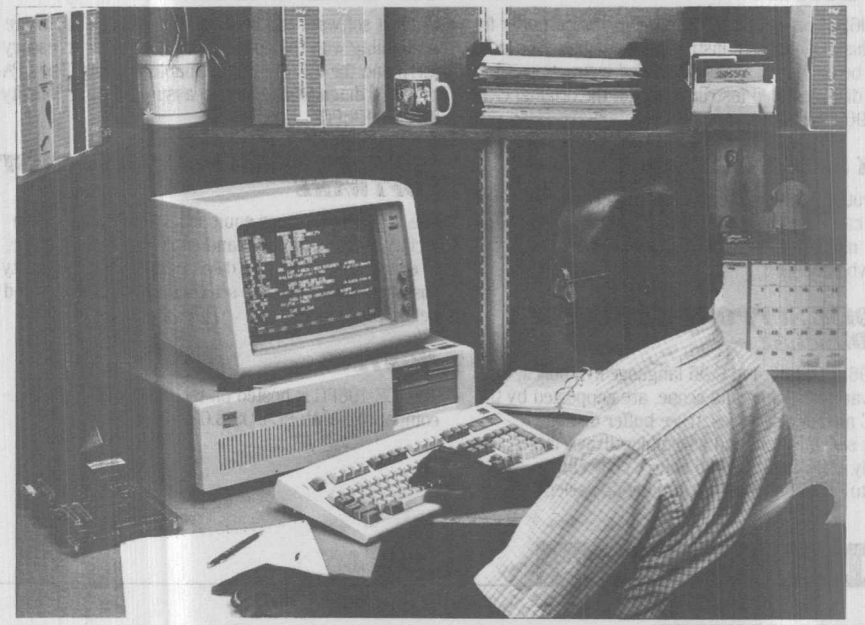




ORDERING INFORMATION

ICE286	ICE-286 NMOS System including ICE S/W packages (Requires DOS 3.XX PC AT with Above Board)
ICE286PAT	ICE-286 NMOS System including ICE S/W Packages and the iPAT system (Requires DOS 3.XX PC AT with Above Board)
D86ASM286NL	286 macro assembler 286 builder/ binder/mapper utilities for DOS 3.XX.
D86C286NL	286 C compiler and run time libraries for DOS 3.XX.
D86PLM286NL	286 PL/M compiler for DOS 3.XX.

REAL-TIME TRANSPARENT 80C196 IN-CIRCUIT EMULATOR



REAL-TIME TRANSPARENT 80C196 IN-CIRCUIT EMULATOR

The ICE-196PC in-circuit emulator delivers real-time high-level debugging capabilities for developing, integrating and testing 80C196-based designs. Operating at the full speed of the 80C196 microcontroller, the ICE-196PC provides precise I/O pin timings and functionality. The ICE-196PC also allows you to develop code before prototype hardware is available. The ICE-196PC in-circuit emulator represents a low-cost development environment for designing real-time microcontroller-based applications with minimal investment in time and resources.

ICE™-196PC IN CIRCUIT EMULATOR FEATURES

- Real-Time Emulation of the 80C196 Microcontroller
- 64K Bytes of Mappable Memory
- 2K-entry Trace Buffer
- 3 Breakpoints or 1 Range Break
- Symbolic Support and Source Code Display
- Standalone Operation
- Versatile and Powerful Host Software
- Hosted On IBM PC XT, AT* or Compatibles With DOS 3.0 or Later

REAL-TIME EMULATION

The ICE-196PC provides real-time emulation with the precise input/output pin timings and functions across the full operating frequencies of the 80C196 microcontroller. The ICE-196PC connects to the intended 80C196 microcontroller socket via a 16″ flex cable, which terminates in a 68-pin PLCC probe.

MAPPABLE MEMORY

The ICE-196PC has 64K bytes (65,536) of zero wait-state memory that can be enabled or mapped as read-only, write-only or read/write in 4K byte increments to simulate the internal (EP)ROM of the 80C196 or external program memory.



intel

*PC XT, AT are trademarks of IBM.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.

© Intel Corporation 1987

AUGUST, 1987
Order Number: 280727-001

TRACE BUFFER

The ICE-196PC contains a 2K (2048) entry trace buffer for keeping a history of actual instruction execution. The trace buffer can be displayed as disassembled instructions or, optionally, disassembled instructions and the original C-96 and PL/M-96 source code.

BREAK SPECIFICATION

Three execution address breakpoints or one range of addresses can be active at any time. The ICE-196PC allows any number of breakpoints to be defined and activated when needed.

SYMBOLIC SUPPORT AND SOURCE CODE DISPLAY

Full ASM-96, PL/M-96 and C-96 language symbolics, including variable typing and scope, are supported by the ICE-196PC memory accesses, trace buffer display, breakpoint specification, and assembler/disassembler. Additionally, C-96 and PL/M-96 source code can be displayed to make development and debug easier.

STANDALONE OPERATION

Product software can be developed prior to hardware availability with the optional Crystal Power Accessory (CPA) and the ICE-196PC mappable memory. The CPA also provides diagnostic testing to assure full functionality of the ICE-196PC.

VERSATILE AND POWERFUL HOST SOFTWARE

The ICE-196PC comes equipped with an on-line help facility, a dynamic command entry and syntax guide, built-in editor, assembler and disassembler, and the ability to customize the command set via literal definitions and debug procedures.

HOSTING

The ICE-196PC is hosted on the IBM PC XT, AT or compatibles with PC-DOS 3.0 or later.

SPECIFICATIONS

HOST REQUIREMENTS

IBM PC XT, AT (or compatible)
- 512K bytes RAM, Hard Disk
- PC-DOS 3.0 or Later
- One Unused Peripheral Slot
- DC Current 2.5A

TARGET INTERFACE BOARD

Length	2.0" (5.1cm)
Height	1.2" (3.0cm)
Width	2.3" (5.8cm)

USER CABLE

Length 15.6" (39.6cm)

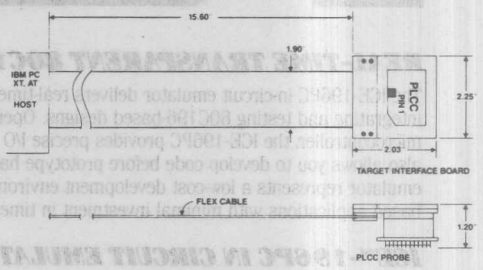
PROBE ELECTRICAL

80C196* plus per pin	50pf loading 5ns propagation delay
Icc (from target system)	50mA @ 12 MHz
Operating Frequency	3.5 to 12 MHz, 12 MHz only with CPA

ENVIRONMENTAL CHARACTERISTICS

Operating Temperature	10°C to 40°C 37.5°F to 104°F
Operating Humidity	Maximum 55% Relative Humidity, non-condensing

*This emulator supports the initial 80C196 microcontroller. The HOLD/HOLDA feature will be supported by a future product.



ORDERING INFORMATION

Order Code	Description
ICE-196PC	Emulation Board, user cable, target interface board (PLCC), host, diagnostic, and tutorial software on 5¼" DOS diskette, and Crystal Power Accessory with power cable
ICE-196PCB	Same as above except does not include Crystal Power Accessory
CPA196	Crystal Power Accessory and power cable only
D86C96NL	C-96 Compiler*
D86PLM96NL	PL/M-96 Compiler*
D86ASM96NL	ASM-96 Assembler*

*Includes: Relocator/Linker, Object-to-hex Converter, Floating Point Arithmetic Library, Librarian

For more information or the number of your nearest sales office call 800-548-4725 (good in the U.S. and Canada).

UNITED STATES, Intel Corporation
3065 Bowers Ave., Santa Clara, CA 95051
Tel: (408) 987-8080